2014-1516, -1530

United States Court of Appeals for the Federal Circuit

SYNOPSYS, INC.,

Appellant,

v.

MENTOR GRAPHICS CORPORATION,

Cross-Appellant.

Appeals from the United States Patent and Trademark Office, Patent Trial and Appeal Board in No. IPR2012-00042.

PRINCIPAL AND RESPONSE BRIEF WITH ADDENDUM OF CROSS-APPELLANT MENTOR GRAPHICS CORPORATION

Robert A. Long, Jr.
Matthew J. Berns
COVINGTON & BURLING LLP
One CityCenter
850 Tenth Street, NW
Washington, DC 20001
Telephone: (202)662-6000

Christopher L. Mckee
Bradley C. Wright
Michael S. Cuviello
BANNER & WITCOFF, LTD.
Suite 1200
1100 13th Street, NW
Washington, DC 20005
Telephone: (202) 824-3000

George A. Riley
Mark E. Miller
O'MELVENY & MYERS LLP
Two Embarcadero Center, 28th Floor
San Francisco, CA 94111
Telephone: (415) 984-8700

Attorneys for Cross-Appellant Mentor Graphics Corporation

CORRECTED

CERTIFICATE OF INTEREST

Counsel for the Cross-Appellant Mentor Graphics Corporation certify the following:

1. The full name of every party or amicus represented by us is Mentor Graphics Corporation.

2. The name of the real party in interest (if the party named in the caption is not the real party in interest) represented by us is: N/A.

3. All parent corporations and any publicly held companies that own 10 percent or more of the stock of the party or amicus curiae represented by us: None.

4. The names of all law firms and the partners or associates that appeared for the party or amicus now represented by us in the trial court or agency or are expected to appear in this court are:

Robert A. Long, Jr.
Matthew J. Berns
COVINGTON & BURLING LLP
One CityCenter
850 Tenth Street, NW
Washington, DC 20001
Telephone: (202) 662-6000

Christopher L. McKee
Bradley C. Wright
Michael S. Cuviello
BANNER & WITCOFF, LTD.
Suite 1200
1100 13th Street, NW
Washington, DC 20005
Telephone: (202) 824-3000

George A. Riley
Mark E. Miller
O'MELVENY & MYERS LLP
Two Embarcadero Center, 28th Floor
San Francisco, CA 94111
Telephone: (415) 984-8700

Dated: December 17, 2014

/s/ Christopher L. McKee
Christopher L. McKee
Attorney for Cross-Appellant Mentor Graphics Corporation

TABLE OF CONTENTS

ADDENDUM

TABLE OF AUTHORITIES

Cases

Statutes

Other Authorities

Federal Rules and Regulations

Legislative Materials

STATEMENT OF RELATED CASES

Mentor Graphics Corporation ("Mentor") adopts the Statement of Related Cases in the principal brief of Synopsys, Inc. In addition, Mentor states as follows:

On October 9, 2014, after Synopsys filed its principal brief, the United States District Court for the Eastern District of Virginia dismissed the Administrative Procedure Act ("APA") suit filed by Synopsys to challenge the policy of the Patent & Trademark Office ("PTO") that allows the Patent Trial and Appeal Board ("Board") to limit the scope of *inter partes* review ("IPR") proceedings to a subset of the claims challenged in the petition for IPR. Synopsys appealed from the district court's order, and the appeal has been docketed in this Court as *Synopsys, Inc. v. Lee*, No. 15-1183.

On October 10, 2014, after Synopsys filed its principal brief, a jury in *Mentor Graphics Corp. v., EVE-USA, Inc. et al.*, 3:10-954-MO (D. Or.) (lead) (consolidated cases: No. 3:12-cv-1500-MO; 3:13-cv-579-MO), rendered a verdict finding Synopsys liable for direct, induced and contributory infringement of each of claims 1, 24 and 26-28 of the '376 patent (A95-128). The jury awarded lost profits damages in excess of $36 million, plus reasonable royalty damages. Mentor has filed a motion for a permanent injunction, which remains pending.

JURISDICTIONAL STATEMENT

Mentor adopts Synopsys' Statement of Jurisdiction but disagrees that the Board had jurisdiction to institute an IPR. *See infra* Argument, § V. Mentor further states that it timely filed its notice of cross-appeal. A856-862; F. R. App. P. 4(a)(3); 37 C.F.R. § 90.3(a)(1).

STATEMENT OF THE ISSUES

I. ISSUES PRESENTED BY SYNOPSYS' APPEAL

1. Whether substantial evidence supports the Board's determination that Synopsys failed to satisfy its burden of proving that claims 1 and 28 of the '376 patent are anticipated by Gregory (A1103-1148).

2. Whether the Board correctly limited the patentability issues addressed in its final written decision ("FWD") to the grounds (and associated claims) on which the Board instituted the *inter partes* review, and whether Synopsys waived its argument to the contrary.

II. ISSUES PRESENTED BY MENTOR'S CROSS-APPEAL

1. Whether the Board erred in refusing to dismiss the IPR as barred under 35 U.S.C. § 315(b), because Emulation and Verification Engineering, S.A. and EVE-USA, Inc. (collectively "EVE") were in privity with Synopsys and real parties in interest, and were served with a complaint alleging infringement of the '376 patent more than a year prior to the filing date of the petition.

2. Whether the Board erred in denying Mentor's contingent motion to amend claims 5, 8, and 9.

STATEMENT OF THE CASE

I. STATUTORY BACKGROUND

In the Leahy-Smith America Invents Act ("AIA"), Pub. L. No. 112-29, 125 Stat. 284 (2011), Congress replaced *inter partes* reexamination of patents with an adjudicative proceeding called *inter partes* review. *See* 35 U.S.C. §§ 311–319.

The AIA encourages the use of *inter partes* review by requiring streamlined proceedings. The Board ordinarily must decide whether to institute review within three months after the patent owner files its preliminary response to the petition for *inter partes* review, *id.* § 314(b), and "issue a final written decision with respect to the patentability of any patent claim challenged by the petitioner and any new claim added under section 316(d)" within one year of its institution decision, *id.* §§ 316(a)(11) , 318(a).[1]

At the same time, the AIA protects patent owners by making it more difficult for petitioners to obtain administrative review and to prevail once an administrative proceeding is instituted. The statute elevates the standard for initiating administrative review, providing that the Board "may not authorize an

[1] Section 316(d)) allows the patent owner, "[d]uring an inter partes review," to file a motion to amend the patent. 35 U.S.C. § 316(d).).

inter partes review to be instituted . . . unless . . . there is a reasonable likelihood that the petitioner would prevail with respect to at least 1 of the claims challenged in the petition." 35 U.S.C. § 315(a) ; *cf.* 35 U.S.C. § 312 (2010) (substantial new question of patentability standard for *inter partes* reexamination). This "elevated threshold[]" is "[a]mong the most important protections for patent owners added by the [statute]." 157 Cong. Rec. S1375 (daily ed. Mar. 8, 2011) (statement of Sen. Kyl).

The AIA also provides that "[a]n inter partes review may not be instituted if," among other things, "the petition requesting the proceeding is filed more than 1 year after the date on which the petitioner, real party in interest, or privy of the petitioner is served with a complaint alleging infringement of the patent." 35 U.S.C. § 315(b). This prohibition protects patent owners from having to defend before the Board patents that have been the subject of litigation between the patent owner and the petitioner or certain non-parties. *See, e.g.*, *Office Patent Trial Practice Guide*, 77 Fed. Reg. 48,756, 48,759 (Aug. 14, 2012) ("OPTPG")(§ 315(b) "seeks . . . to prevent parties from having a 'second bite at the apple.'").

The AIA shifts to the IPR petitioner "the burden of proving a proposition of unpatentability by a preponderance of the evidence." 35 U.S.C. § 316(e); *see also* 157 Cong. Rec. S1376 (daily ed. Mar. 8, 2011) (statement of Sen. Kyl) ("One

important structural change . . . is that . . . the petitioner, rather than the Office, bears the burden of showing unpatentability.").

An IPR is restricted to arguments and evidence proffered:

1) by the petitioner in the petition (35 U.S.C. § 312(a)(3)); and

2) by the patent owner in the optional patent owner preliminary response and in the patent owner response (35 U.S.C. §§ 313, 316(a)(8); 37 C.F.R. § 42.107(c)).

While the petitioner may file a reply brief rebutting arguments raised in the patent owner's response, no additional petitioner arguments or evidence is permitted to cure evidentiary or other deficiencies in the petition. *See* 35 U.S.C. § 316(a)(13) (providing only for "written comments" in petitioner reply); 35 U.S.C. § 312(a)(a "petition filed under section 311 may be considered only if - [the petition meets the requirements of § 312(a)(1-5)]." (underscore added)); 37 C.F.R. § 42.23(b). These petition requirements include identifying "with particularity . . . the grounds on which the challenge to each claim is based, and the evidence that supports the grounds for the challenge to each claim." 35 U.S.C. § 312(a)(3); *See also*, 37 C.F.R. § 42.104(b)(2-5)(petition required to identify, e.g., where each element of the claim is found in the prior art, and any evidence relied upon and its relevance). The Board may exclude or give no weight to evidence

where the petitioner fails to state its relevance or identify specific portions of the evidence that support the challenge. 37 C.F.R. § 42.104(b)(5).

These limitations further the Board's statutory mandate to consider "the efficient administration of the Office [] and the ability of the Office to timely complete proceedings instituted." 35 U.S.C. § 316(b). Because the patent owner is provided no opportunity to address arguments that are not presented in the petition, these limitations also provide fundamental fairness and procedural due process.

II. *INTER PARTES* REVIEW OF THE '376 PATENT

On September 26, 2012, Synopsys filed a petition for *inter partes* review for claims 1-15 and 20-33 of the '376 patent. Synopsys challenged the patentability of these claims on multiple grounds, including anticipation by the Gregory patent. A135-199.

Mentor filed a Preliminary Response (A237-289) defending the claims on the merits and arguing that the Board lacked authority to institute the review under 35 U.S.C. § 315(b) based on (a) Mentor's 2006 infringement action asserting the '376 Patent against EVE, and (b) Synopsys' acquisition of EVE – creating privity between EVE and Synopsys.

On February 22, 2013, the Board entered an order instituting the *inter partes* review. A1-41. The Board granted the petition only as to claims 1-9, 11, 28, and

29, and limited the trial to the issue of anticipation of those claims by the Gregory patent. A40. The Board denied the petition as to claims 10, 12-15, 20-27, and 30-33 on the ground that Synopsys had "not demonstrated a reasonable likelihood of prevailing on its challenge to the patentability of [these claims]." A40. The Board denied the petition as to claims 1-9, 11, 28, and 29 with respect to references other than Gregory, and with respect to Synopsys' contention that Gregory rendered those claims obvious. A2, A14. The Board also rejected Mentor's argument that it lacked authority to conduct the *inter partes* review under 35 U.S.C. § 315(b), A15-18, and subsequently denied Mentor's motions for reconsideration, A371-377, and for discovery specifically directed to showing that EVE is a real party in interest and that Synopsys and EVE were in privity. A378-385.

Following institution of the IPR trial, Mentor filed a Patent Owner Response ("PO Resp.") (A398-464) together with a supporting declaration of Dr. Majid Sarrafzadeh (A1864-1998). Synopsys filed a Reply (A549-569) but offered no expert testimony or other evidence. Mentor also filed a contingent motion to amend the patent claims, pursuant to 35 U.S.C. § 316(d) and 37 C.F.R. § 42.121, which Synopsys opposed.

On February 19, 2014, after a hearing, the Board issued a FWD. A42-94. The Board concluded that Synopsys had not proven that Gregory anticipated claims 1-4, 6, 7, 11 or 28-29, but that claims 5, 8 and 9 are anticipated by Gregory.

The Board also denied Mentor's contingent motion to amend. Finally, the Board again addressed Mentor's argument that § 315(b) deprived the Board of authority to conduct the *inter partes* review, rejecting that argument for reasons different than those the Board had previously offered.

Synopsys appealed, challenging the Board's patentability determination only with respect to claims 1 and 28. Mentor cross-appealed.

STATEMENT OF THE FACTS

I. FACTS PERTINENT TO THE § 315(B) BAR

In 2000, after assigning the '376 patent to Mentor, Luc M. Burgun, one of the named inventors, left Mentor and founded EVE, where he acted as President and CEO. A346, A1842-1843, A1862-1863. In 2006, Mentor served EVE with a complaint alleging that EVE's ZeBu verification system infringed the '376 patent and two other Mentor patents (the "2006 Action"). A1562-1568. The 2006 Action was dismissed in December 2006 pursuant to a settlement agreement. A1572. On September 27, 2012—one day after Synopsys filed its petition for IPR—Synopsys and EVE filed a joint action for a declaratory judgment of invalidity and non-infringement concerning the '376 patent (A1583, A1573-A1588), and by October 4, 2012, just days after Synopsys filed its IPR petition, Emulation Verification and Engineering S.A. became a wholly-owned subsidiary of Synopsys. A1590, A1592.

II. TECHNOLOGY BACKGROUND AND THE DIFFERENT PROBLEMS ADDRESSED BY THE '376 PATENT AND GREGORY

The '376 patent addresses shortcomings in previously known techniques for verifying electrical circuits intended to be implemented in an integrated circuit. Circuit designers design integrated circuits, often called chips, using Hardware Description Languages ("HDL"). Register Transfer Level ("RTL") languages were and are the preferred HDL languages used for circuit design. The invention claimed in the '376 Patent involves the use of instrumentation signals to provide visibility into RTL circuit designs in order to identify and locate errors in the designs. The process of identifying and locating problems in the design is called "debugging." Different types of debugging are carried out at different stages of the design process. Emulators allow verification engineers to observe the operation of portions of a circuit design under test that cannot be observed during operation of a chip implementing the same design. A crucial element of modern circuit design testing is the ability to debug a design at the same level of abstraction in which the code describing the circuit design is written. Circuit designs described using RTL are at a higher level of abstraction than a logic gate level or transistor level description of a circuit design. *See* A119 ('376 patent, 1:15-2:24).

More specifically, the process of designing a modern-day integrated circuit typically begins with a high-level algorithmic description of the desired operations of the circuit. From this, a separate RTL model written in a Hardware Description

Language (HDL) can be developed, which adds a level of detail by scheduling those operations according to a clock signal. A1599, A1616.

A process known as "synthesis" is then used to create, from the abstract description of operations in the RTL/HDL model, a corresponding description of the circuit's structure that carries out the operations (a specification of logic level gates corresponding to the physical circuit components and their electrical interconnections (nets)). This low-level structural specification is known as a "gate-level netlist." *See* A119 ('376 patent, 1:15-2:24).

Synopsys would have the Court believe that Gregory and the '376 patent address the same problem and provide the same solution. They do not. These two patents address distinctly different phases and aspects of digital microcircuit design.

Referring to Gregory's prior art Fig. 1 (reproduced below), illustrated is "an overview of the conventional process for designing and debugging circuits specified with [an HDL]" (A1139 at 4:20-23).

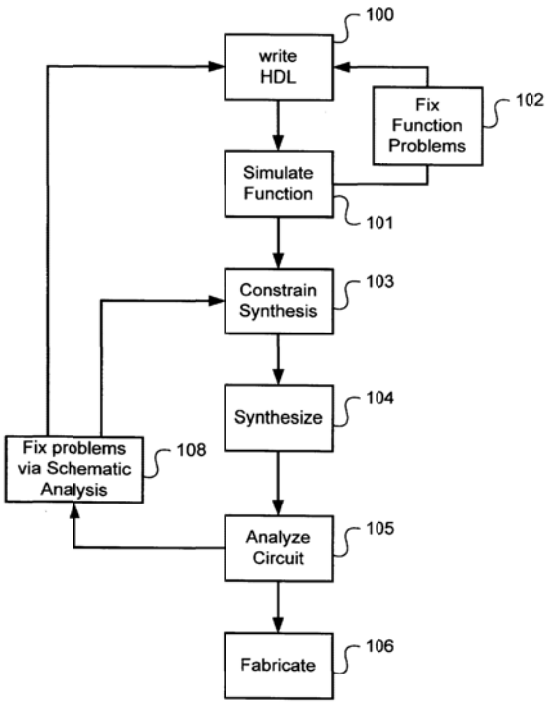


Figure 1

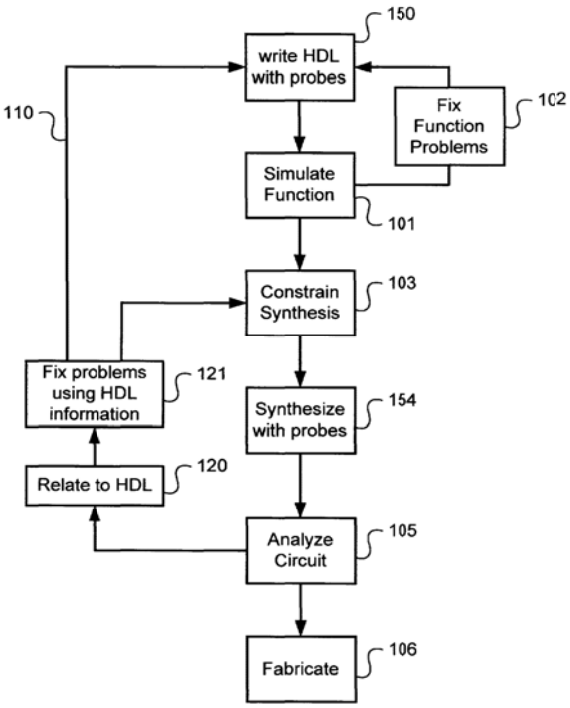


Figure 3

Gregory's Prior Art Fig. 1 **Gregory's Fig. 3**

The steps of the Fig. 1 process are explained at A1139 (Gregory, 4:41-6:43), in which a designer describes his circuit design functionally in HDL, and then goes through different debugging and synthesis steps to generate a gate-level hardware description suitable for fabrication. Gregory's disclosed techniques are aimed at facilitating the "debugging" associated with the physical constraints-related "Analyze Circuit" step 105, which is shown by its modification to the Fig. 1 process in Fig. 3. A1140 (6:19:23). In Fig. 3, Gregory adds block 120, which relates "analysis information" generated by step 105 about physical constraints

(e.g., speed and size) of hardware gates back to related functional description in HDL code that has been identified with probe statements. A1143 (11:29-32).

The ‘376 patent techniques pertain to functional “debugging” associated with a gate-level simulation step (not disclosed by Gregory), where the gate-level simulation can be performed in, for example, a hardware emulator. *See, e.g*. A120-121 (’376 patent, 4:58-5:8). Fig. 1 of Gregory (A1105) includes a “simulate function” step 101, but this functional simulation is carried out at the HDL level (pre-synthesis), rather than at the gate-level (post-synthesis) as described in the ‘376 patent. In contrast to the ‘376 patent, Gregory omits any functional simulation step after synthesis. It is instructive to constrast the pre-synthesis, HDL level of Gregory’s “Simulate Function” step 101 shown in the design flow of Gregory’s Fig. 1 (above) to the post-synthesis, “Gate-Level Simulation” step 160 shown in the design flow of Fig. 1 of the ‘376 patent (reproduced below):

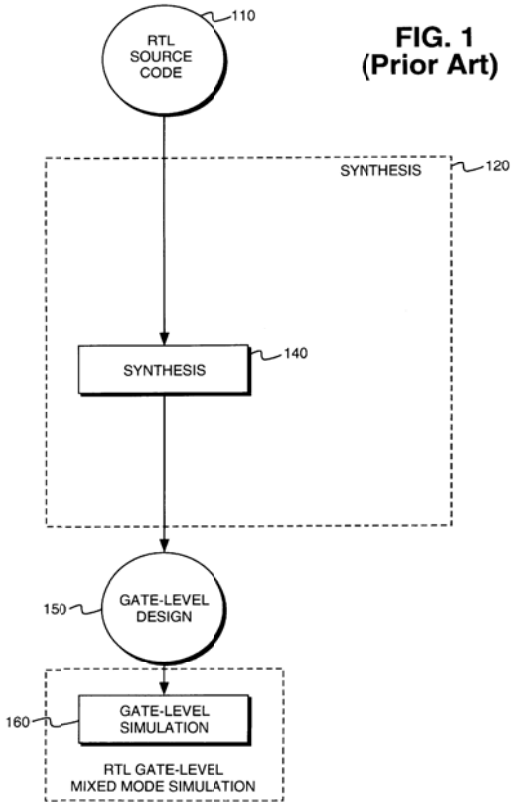


FIG. 1 (Prior Art)

The '376 patent discloses post-synthesis simulation, whereas Gregory discloses functional simulation preceding the synthesis and optimization/verification stages during which Gregory's "probe statements" are used (and which Synopsys alleges result in instrumentation signals as claimed). A1139 (4:55-62).

The purpose of functional "debugging" is to verify whether the logic circuit operates in the manner intended, i.e., that it provides the expected outputs for a given set of inputs, and to correct any design errors in this regard. Functional verification involves actually running (executing) the circuit and tracking execution flows. One reason the '376 patent performs functional verification with a gate-level design (as opposed the RTL design as in Gregory) is so the design can

be implemented in a hardware emulator for carrying out functional verification at a much greater speed than can be attained with software simulation at the RTL/HDL level. This is explained in the '376 patent background section (A119 (1:55-67)). *See also* A1871-1872 (Sarrafzadeh Decl., ¶ 14).

Synopsys conflates the functional debugging addressed by the '376 patent with Gregory's physical constraint "debugging." Physical constraint debugging brings a synthesized gate-level circuit design within specified physical constraints including circuit timing and area requirements (for example, how fast the signals need to travel along their paths and how small the circuit size must be). Gregory addresses physical constraint debugging which occurs late in the design process, after correct functionality of the circuit is verified by functional simulation. Referring again to Gregory's Fig. 1 (reproduced above), Gregory's functional debugging occurs in the "Analyze Circuit" step 105.

The objective of this later, physical phase of "debugging" is to transform a functionally verified circuit design into a functionally equivalent "optimized" design better suited for realization on a microchip. Dr. Sarrafzadeh provided testimony on this point. A1871, ¶ 13. This second type of physical "debugging" does not involve running (i.e., executing) the circuit design, as during functional simulation. Rather, in this stage, software tools make computations based on the specified circuit design and information concerning critical physical parameters,

and assist the designer with identifying physical constraint problems necessitating design changes. *See, e.g.,* A1139 (4:60-63):

> If the designer believes that the described circuit provides the correct function, then designer then specifies constraints for the synthesis process 103, e.g., maximum clocking periods, total circuit area, and maximum power. . . .

SUMMARY OF THE ARGUMENT

I. SYNOPSYS' APPEAL

A. The Board's Determination Upholding Claims 1 And 28 Is Supported By Substantial Evidence

The Court should defer to the Board's finding that Synopsys failed to prove by a preponderance of the evidence that Gregory teaches the "instrumentation signal" indicative of an "execution status" as required by claims 1 and 28. Regarding the alleged anticipation of claims 1 and 28, Synopsys asserted in its petition that the "tempout" signal in Fig. 9 (with respect to claim 1) and the "tempout" signals in Fig. 18 (with respect to claim 28) disclose an "instrumentation signal" indicative of an "execution status." The testimony of Mentor's expert Dr. Majid Sarrafzadeh (which Synopsys offered no evidence to rebut) provides substantial evidence to support the Board's finding that Synopsys failed to show Gregory's "tempout" signals indicate "execution status."

Attempting to rehabilitate its flawed Petition, Synopsys raised new theories of anticipation in its reply (A549-569) to Mentor's Patent Owner Response (A398-

464). The Board correctly recognized that these new theories were barred as untimely and unsubstantiated in any event. A74 (FWD) (citing 37 C.F.R. § 42.23(b); *see also* 35 U.S.C. § 312(a)(3).

Synopsys now seeks a third bite at the apple by advancing on appeal more new anticipation arguments and theories. These belated arguments are waived and meritless.

B. The Board Is Not Required To Address Patent Claims That Are Outside The Scope Of The Inter Partes Review

Synopsys waived its challenge to the Board's FWD under § 318(a) by failing to raise any objection on this particular issue during the IPR proceedings. Further, the PTO has correctly construed the AIA not to require the Board to invest scarce resources on claims for which a petitioner has not demonstrated a reasonable likelihood of success. This ground for Synopsys' challenge to the Board's FWD should be rejected by this Court.

II. MENTOR GRAPHICS' CROSS-APPEAL

A. The IPR Should Have Been Dismissed Under 35 U.S.C. § 315(b)

The Board lacked authority to review the '376 patent based on Synopsys' Petition. The AIA provides that "[a]n inter partes review may not be instituted if the petition requesting the proceeding is filed more than 1 year after the date on which the petitioner, real party in interest, or privy of the petitioner is served with a complaint alleging infringement of the patent." 35 U.S.C. § 315(b). Synopsys

filed the petition more than one year after EVE was served with a complaint alleging infringement of the '376 patent. EVE was both a privy of Synopsys and a real party in interest. The IPR is therefore barred under § 315(b).

B. The Board Erred In Denying Mentor Graphics' Motion To Amend

The Board's *Idle Free* standard, applied in denying Mentor's Motion to Amend (A494-514), impermissibly shifts the burden of proving patentability of the proposed substitute claims from Synopsys to Mentor, contrary to 35 U.S.C. § 316(e). The Board also abused its discretion by unreasonably imposing on Mentor the need to show "general patentability over prior art." A88. This standard, coupled with the Board's procedural constraints, effectively denied Mentor a reasonable opportunity to amend the claims.

ARGUMENT

I. STANDARD OF REVIEW

Mentor agrees with Synopsys (Br. 33) that the Board's anticipation determinations are reviewed under the deferential "substantial evidence" standard. "A finding is supported by substantial evidence if a reasonable mind might accept the evidence to support the finding." *In re Antor Media Corp.*, 689 F.3d 1282, 1287 (Fed. Cir. 2012).

The Court must set aside the Board's denial of Mentor's motion to amend the patent if that decision was "arbitrary, capricious, an abuse of discretion, or otherwise not in accordance with law." 35 U.S.C. § 706(2)(A).

In reviewing Synopsys' challenge to the interpretation of the AIA codified by the PTO in 37 C.F.R. § 42.108(a), the Court must apply the deferential framework of *Chevron, U.S.A., Inc. v. Natural Res. Def. Council, Inc.* 467 U.S. 837 (1984)). *See Cooper Techs. Co. v. Dudas*, 536 F.3d 1330, 1335-38 (Fed. Cir. 2008). The Board is entitled to no deference, however, with respect to statutory interpretations adopted in individual cases, such as its interpretation of § 315(b) in this *inter partes* review. *See In re Swanson*, 540 F.3d 1368, 1374 n.3 (Fed. Cir. 2008).

II. THE BOARD'S DETERMINATION UPHOLDING CLAIMS 1 AND 28 IS SUPPORTED BY SUBSTANTIAL EVIDENCE

A. The Board's Proper Constructions Of "Instrumentation Signal" And "Execution Status"

The Board construed two claim terms: "instrumentation signal" and "execution status." The Board agreed with Mentor, and Dr. Sarrafzadeh, that the "instrumentation" part of "instrumentation signal" generally means "devices or instructions installed or inserted into hardware or software to monitor the operation of a system or component" or "the insertion of additional code into [a] program in order to collect information about program behaviors during program execution."

A60-61 (FWD). The Board also concluded that "instrumentation signal" does not exclude "creation by preserving already existing circuit components," stating:

> that the broadest reasonable construction of 'instrumentation signal,' in light of the '376 patent specification, at least includes an output signal created during synthesis of RTL source code by inserting additional code into a program that indicates whether the corresponding RTL source code statement is active.

A67-68.

The Board determined "that the broadest reasonable construction of 'execution status' in light of the '376 patent specification is information regarding whether a particular HDL instruction has been performed." A68. Synopsys has not challenged the Board's construction of "instrumentation signal" or "execution status." Neither does Mentor.

B. Substantial Evidence Supports The Board's Determination That Synopsys Did Not Prove Gregory To Be Anticipatory Of Claims 1 And 28

1) The Board Reasonably Dismissed The Argument Presented In Synopsys' Petition

Synopsys' Petition presented the argument that Gregory anticipates claim 1 only in claim chart form. A169-170. It did the same with respect to claim 28. On appeal, to demonstrate Gregory's alleged showing of the "instrumentation signal" limitation, Synopsys simply refers the reader to "the analysis of claim 1 for general discussion of using probes to identify or create instrumentation signals." A180-181.

Claim 1 requires an instrumentation signal indicative of "an execution status of the at least one statement." Synopsys alleged in the Petition that:

> . . . synthesizing the FIG. 8 VHDL code produces the gate-level circuit illustrated in FIG. 9, which includes the instrumentation signal temp-out, which indicates an execution status of the instrumented statement.

A170. Claim 28 includes a similar "execution status" limitation, which the Petition failed to address. Instead, the Petition merely contended that Gregory's block probe methodology results in the circuit of Fig. 18 including the signals "temp_out," which Synopsys contended are instrumentation signals. A181.

As the Board recognized, the only argument of anticipation raised in the Petition was that Gregory's artificial inclusion of "tempout" in a synthesized gate level design satisfies the claims 1 and 28 limitations of an "instrumentation signal" that is "indicative of an execution status" of the at least one statement. A169 ("in its petition, the only element Synopsys describes with particularity as disclosing this particular limitation of claim 1 is the 'tempout' signal of Fig. 9"). But, the Board determined that Gregory's "tempout" had not been shown to satisfy the execution status limitation of claims 1 and 28. A171.

Mentor and its expert showed that "tempout" is not "indicative of an execution status." As the Board explained:

> We give significant weight to the testimony of Mentor Graphics's expert, Dr. Sarrafzadeh, who persuasively explains that Gregory does not disclose each and every

> element of the claims. Synopsys does not provide sufficient evidence to rebut this testimony.

A69. The Board found persuasive Dr. Sarrafzadeh's testimony "that in Fig. 9, the result of the 'tempout' signal is not indicative of the execution status of the HDL statement identified in Fig. 8. Ex. 2027 ¶73." *Id.* *See also,* A72-73 (giving substantial weight to Dr. Sarrafzadeh's testimony (A1905-1907 (¶¶ 73-78) over "Synopsys' unsupported argument to the contrary."). And, the Board recognized the difference between the functional debugging aspect of microcircuit design addressed by the '376 patent and the physical constraint (size and timing) debugging aspect of microcircuit design addressed by Gregory. With regard to this distinction, the Board concluded:

> Synopsys does not point to any disclosure in Gregory that "analysis information" includes "execution status." PO Resp. at 47. In fact, when Gregory discusses "analysis," it generally refers to characteristics of the circuit, such as timing and power, which are not related to "execution status" as we have construed that term.

A73. Synopsys has not presented any evidence that refutes these points.

The Board further considered and rejected Synopsys' counter-argument presented in its Reply to the Patent Owner Response based on an alternate cross-reference database embodiment described in the '376 patent.[2] A562-563.

[2] The Board, in addressing Synopsys' Reply Brief, refers to the "alternate embodiment also described in Gregory." A562-563(citing Reply 10-11). From

According to Synopsys "because execution status of every branch can be verified implicitly and not every branch need be instrumented, Mentor's argument is incorrect. [Reply] at 11 (citing Ex. 1001, col. 12, ll. 33-38)." A72 (FWD). The Board found this argument unpersuasive, noting that Gregory does not explicitly disclose this and "Synopsys [] does not point to any expert testimony to support these statements." *Id.*

2) The Board Correctly Rejected Arguments Synopsys Presented For the First Time In Its Reply As Waived And On The Merits

Synopsys' Brief (at pages 64-68) further presses a new theory of anticipation first presented in its Reply: that "execution status is disclosed through the output of the AND gate (232)" in the circuit of Fig. 9 in Gregory. Br. 64.

With respect to the Board's holding, Synopsys makes two points: 1) the argument was not waived, and 2) the Board failed to consider this argument on its merits in the FWD (Br. 65).

The Board correctly rejected Synopsys' new Reply argument as untimely, noting "the argument is presented for the first time in Synopsys' reply and is not responsive to arguments made in Mentor's response." A74; *See* 37 C.F.R. § 42.23. Synopsys attempts to portray the belated presentation of this alternative theory as

---

review of the Reply, it is clear the Board meant to refer to the alternate embodiment of the '376 patent, which is also discussed on pages A64-66.

responsive to arguments presented by Mentor in its Patent Owner Response. Br. 66-67. However, Mentor's Response addressed directly the only contention raised in the Petition, i.e., that "tempout" of Fig. 9 was an instrumentation signal indicative of execution status. Synopsys' Reply argument that some other signal in Fig. 9 satisfies the claim language plainly represents a shift to a new alternative theory of anticipation, **not** responsive argument supportive of the original theory of anticipation that Synopsys put forward in its Petition.

Synopsys asserts that Mentor's argument challenging Gregory's teaching of "execution status" was not presented in Mentor's "Preliminary Response." Earlier presentation of this argument in the optional Preliminary Response would have made no difference. This is so, because other than requesting rehearing of the institution decision (37 C.F.R. § 42.71(d)), Synopsys' only opportunity to respond as a matter of right was with its reply to the Patent Owner Response. And, that reply "may only respond to arguments raised in the corresponding . . . patent owner response." 37 C.F.R. § 42.23(b). Arguments that go beyond this are properly disregarded by the Board. *Id.*; *See also*, OPTPG, 77 Fed. Reg. at 48,767 (new issues or evidence in a reply will not be considered).

Such untimely arguments raising issues on which Mentor had no opportunity to introduce evidence are statutorily prohibited (35 U.S.C. §312(a)(3)) and should

be deemed waived on appeal. *See Johns Hopkins University v. CellPro, Inc.*, 152 F.3d 1342, 1362 (Fed Cir. 1998).

The Board also found that Synopsys' argument regarding signals other than "tempout" (e.g., the output of the AND gate 232)[3] as being an instrumentation signal to be unsupported by evidence. A74 ("Moreover, Synopsys does not point to any evidence or persuasive argument to explain how B and C disclose the claimed instrumentation signal.") Synopsys' Appeal Brief fails to address this finding, but instead focuses solely on the distinct "execution status" limitation.

The Board's finding on the merits is correct. Substantial evidence supports the conclusion that the output of the AND gate is (like "tempout") not an "output signal" as would be required under the Board's undisputed claim construction of "instrumentation signal." A67.

C. The Board Applied The Correct Standard For Anticipation

The Board analyzed Synopsys' anticipation contentions under the correct standard. S*ee* A68-69 (citing *Net MoneyIN, Inc. v. VeriSign, Inc.*, 545 F.3d 1359, 1369 (Fed. Cir. 2008)). Synopsys misconstrues the FWD by focusing only on a portion of this analysis to assert that the Board improperly applied an *ipsissimus verbis* anticipation test and did not properly consider the possibility of inherent

[3] The AND gate 232 performs the logical combination of signals B and C, and thus, reference to "B and C" is equivalent to reference to the output of the AND gate.

anticipation. Br. 31 and 47-50.

Synopsys argues on Appeal that the Board made an error in law by requiring Gregory to explicitly disclose "execution status," and thus, presumably disregarding that anticipation may be found by inherent or implicit disclosure of the claimed feature. *See* Br. 31-32, 47-50. Synopsys also argues that the Board erred based on its "insistence on expert testimony." Br. 31 and 61-62.

The Board held that Synopsys had provided insufficient evidence to support its theory of anticipation (*see* A69, A72-73). This was not a "requirement that the prior art must 'explicitly' disclose a claim limitation" as argued by Synopsys. Br. 31. Rather, it simply reflects the Board's proper determination that "execution status" was not disclosed in Gregory and that Synopsys had failed to meet its burden of proving the presence of the feature in Gregory.

For the things Gregory does not disclose explicitly but which Synopsys says are nonetheless implicit or inherent, a fact issue is raised as to whether one of ordinary skill in the art would understand the inherent disclosure to be necessarily present. In general, proof of inherent disclosure may require extrinsic evidence (i.e., evidence outside of reference) to make clear that the missing descriptive matter is necessarily present, and that it would be so recognized by persons of ordinary skill. *See In re Robertson*, 169 F.3d at 1950-55 (citing *Continental Can*

Co. v. Monsanto Co., 948 F.2d 1264, 1268 (Fed.Cir. 1991); *see also Transclean Corp. v. Bridgewood Serv. Inc.*, 290 F.3d 1364, 1373 (Fed. Cir. 2002).

In contending (on appeal) that features allegedly implicit in Gregory anticipate claims 1 and 28, Synopsys provides only attorney argument, which does not substitute for evidence. *See Crown Operations Int'l, Ltd. v. Solutia Inc.*, 289 F.3d 1367, 1375 (Fed. Cir. 2002) (party's assumptions and own contentions were insufficient to show inherency). In the absence of expert testimony or other sufficient evidence to show the alleged features, which were not explicitly disclosed in Gregory, the Board properly found Synopsys' arguments insufficient to meet its burden of proving anticipation by a preponderance of the evidence (35 U.S.C. § 316(e)).

Additionally, the Court should not consider arguments and evidence of anticipation that were not timely raised. Had Synopsys desired to put forth an inherency theory of anticipation by Gregory, it was obligated to have stated its theory and identified the supporting evidence "with particularity" in the Petition. 35 U.S.C. § 315(a)(3); 37 C.F.R. § 42.104(b).

III. SYNOPSYS' ANTICIPATION ARGUMENTS PRESENTED FOR THE FIRST TIME ON APPEAL SHOULD BE DEEMED WAIVED AND ARE WITHOUT MERIT

A. The Court Should Deem As Waived Synopsys' Anticipation Arguments Presented for the First Time On Appeal

Synopsys presents new anticipation arguments and theories on appeal. Such arguments are statutorily prohibited (35 U.S.C. § 312(a)(3)) and should be deemed waived. *CellPro*, 152 F.3d at 1362. This rule addressed by *CellPro* extends to new theories advanced in support of asserted invalidity grounds, *id*. at 1361-62, and ensures "that litigants may not be surprised on appeal by final decision there of issues upon which they have had no opportunity to introduce evidence," *id*. at 1362.

In stark contrast to its sparse presentation in the Petition (*see supra* Argument, § II.B.1)), Synopsys presents in its Brief (1) a new argument (at 59-61) regarding a newly presented hypothetical scenario purporting to show that "tempout" of Gregory's Figs. 8-9 is indicative of "execution status"; and (2) a new argument (at 54-58) regarding a newly presented hypothetical similarly purporting to show that the "tempouts" of Gregory's embodiment of FIGS. 16 and 18 satisfy the "execution status" limitation. These arguments go well beyond any argument that Synopsys presented to the Board, and thus are not properly presented on appeal.

The basis for Synopsys' appeal seems to be that the Board declined to fill the gaps in and provide the evidence missing from Synopsys' Petition. *See* Br. 62

("The Board is comprised of judges with technical training . . . and is constituted to evaluate technical arguments about technical material. . . . The Board's job is to assess the prior art and to determine who has the better arguments, not to count who has more experts.").

An IPR is an adjudicative trial that, once instituted, puts the burden of proving unpatentability on the Petitioner. 35 U.S.C. § 316(c). It is not the role of the Board to perform an "examination" of the patent. Synopsys' opportunity, and its obligation, was to present its complete case in its Petition, not later, and especially not for the first time on appeal. 37 C.F.R. §§ 42.22 and 42.104.

B. Synopsys' New Appeal Brief Theories Do Not Prove Gregory Discloses an Instrumentation Signal Indicative Of "Execution Status"

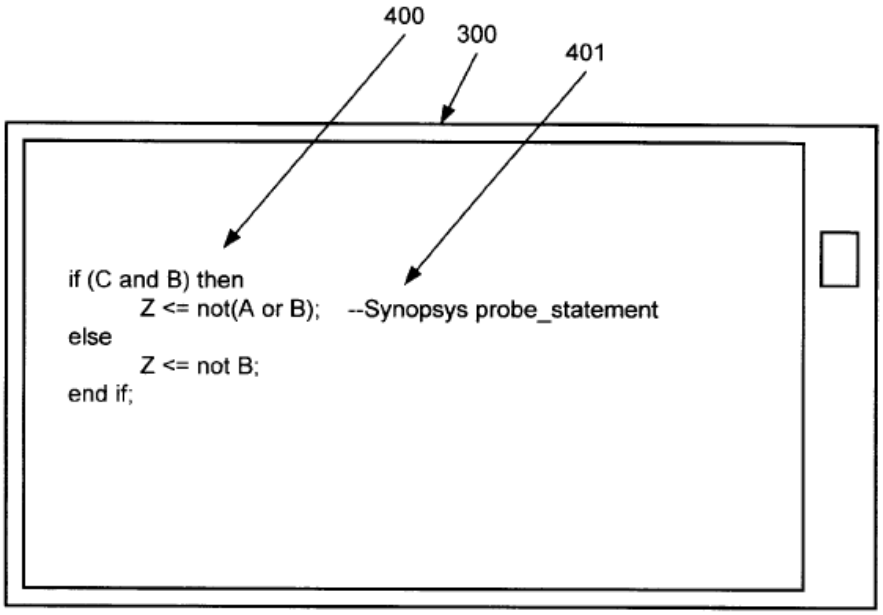
Synopsys presents for the first time on appeal new theories of anticipation. These new theories are included in the Appeal Brief on pages 54-58 with respect to Figures 16 and 18 of Gregory and on pages 59-61with respect to Figures 8 and 9. Synopsys has waived all such arguments.

Even if these new arguments are considered, all of the new appeal arguments are based on speculation about how someone might have functionally simulated the circuits presented in Gregory and/or based on assumptions about the higher level circuits that contain the circuit fragments illustrated in figures 8, 9, 16, and 18 of Gregory. These new purported facts are not disclosed in Gregory (explicitly or

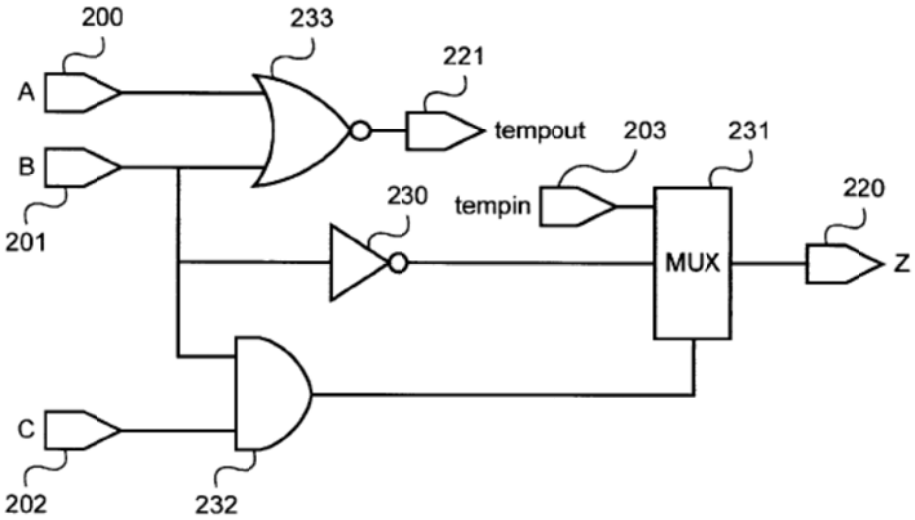
inherently). Synopsys' reliance on purported facts that are not disclosed in Gregory demonstrate that Gregory does not anticipate claim 1 or claim 28.

1) Synopsys' new theory regarding Figures 8 and 9 of Gregory

Synopsys' new arguments (at 59-61) with respect to figures 8 and 9 fail to show any of the alleged deficiencies in the testimony of Dr. Sarrafzadeh and the holding of the Board. Synopsys' Petition identified "tempout" of Fig. 9 as an instrumentation signal, and in response, Mentor, through its expert, demonstrated how "tempout" did not indicate execution status. A170 (Petition); A449-453 (PO Resp.). Figures 8 and 9 of Gregory are reproduced below.



Gregory, Fig 8.



Gregory, Fig. 9

The Board gave substantial weight to Dr. Sarrafzadeh's testimony, and noted that Synopsys' arguments to the contrary were unsupported and not persuasive. A72-73. In the FWD, the Board summarized Dr. Sarrafzadeh's opinion as follows:

> Dr. Sarrafzadeh testifies that in Figure 9, the result of the "tempout" signal is not indicative of the execution status of the HDL statement identified in Figure 8. Ex. 2027 ¶ 73. . . . "tempout" reflects the result of "not (A or B)," while the HDL statement identified in Figure 8 is "Z<=not (A or B)," executed conditionally based on whether the "if" condition (if (C and B) then)) is true. *Id.* at ¶¶ 73-74. The execution status of the HDL statement thus depends on whether the "if" condition is true or not, but "tempout" does not indicate this information. *Id.*

A72.

Synopsys agrees with this analysis, at least in some circumstances:

> ***That may be in the circumstance where "tempout" has a value of 0.*** But the Board's broad conclusion—that "tempout" "does not indicate" "whether the 'if' condition [if (C and B)] is true"—is simply and flatly mistaken. As

> explained immediately above, if the value of "tempout" is 1, then the value of B must be 0. . . .. Knowing that the value of B is 0 means knowing that the condition "if (C and B)" is not true, that the "else" line in the source code of Figure 8 would execute, and that Z equals 1 because "Z <= not(B)."

Br. 60-61 (emphasis added).[4]

This new theory amounts to an argument that Fig. 9, under certain functional simulation conditions, results in "tempout" not indicating "execution status" (as Dr. Sarrafzadeh showed), and under other functional simulation conditions results in "tempout" indicating execution status (which Synopsys attempts to show).

Such an argument fails for many reasons. Starting from the proposition that "'execution status' . . . is information regarding whether particular HDL statement has been executed" (A68), the record is clear and the Board specifically found that Gregory never *explicitly* discloses or requires any ***functional simulation*** of the gate level netlists such as the one represented in Fig. 9. *See* A73 (citing A450).

Absent explicit disclosure of functionally simulating the gate level design of Fig. 9, Synopsys belatedly attempts to prove inherent anticipation by presenting

[4] Synopsys' suggestion that Mentor and the Board should have foreseen and addressed this anticipation argument, ***never raised by Synopsys in the IPR*** (Br. 61-62), runs counter to the statutory requirement that Synopsys put forth its contentions in the Petition (35 U.S.C. § 312(a)) and its burden to prove anticipation by a preponderance of the evidence (35 U.S.C § 316(e)). *See supra* Argument III.A.

functional simulation scenarios in which the possibility exists for "tempout" of Fig. 9 to indicate "execution status." The mere possibility of such a scenario is insufficient; to prove inherent anticipation, Synopsys would have had to provide some evidence that made clear that the missing descriptive matter was *necessarily present* in the embodiments described in Gregory. *See In re Robertson* 169 F.3d 743, 745 (Fed. Cir. 1999).

The flow diagrams in Fig. 1 (*see supra* Statement Of The Facts § II) demonstrate that Gregory only discloses performing functional simulation of the HDL model (step 101) before synthesis, while a different "circuit analysis" (step 105) is performed on the gate level design after synthesis. *See* A1139 (4:55-62) (functional simulation executes the HDL, and gate-level circuits are only generated ***after*** the correct function is verified).

This description of Fig. 1 expressly indicates that functional simulation of the gate level design is *not present* in Gregory, which is opposite to the proposition Synopsys seeks to prove with its additional arguments. Synopsys has never pointed to any disclosure in Gregory or any other evidence that indicates otherwise.

Assuming, *arguendo,* that one were to functionally simulate the gate level design of Fig. 9 in Gregory, Synopsys' argument relies on the additional false assumption that the circuit would necessarily have been simulated under the precise input conditions proposed by Synopsys, i.e., that inputs "A and B must

both have a value of 0." Br. 60. Synopsys concedes that there are certain other functional simulation conditions under which "tempout" does not indicate "execution status." (*Id*. at 60-61 ("*That may be in the circumstance where "tempout" has a value of 0."*).

There is no evidence in the record that Synopsys' proposed functional simulation conditions would *necessarily* be performed instead of or in addition to the functional simulation condition in which "tempout" (as Synopsys admits) does not indicate execution status. "Inherency [] may not be established by probabilities or possibilities. The mere fact that a certain thing may result from a given set of circumstances is insufficient." *In re Oelrich*, 666 F. 2d 578, 581 (C.C.P.A 1981); s*ee also, Mehl/Biophile Intl. Corp., v. Milgraum*, 192 F.3d 1362, 1365 (Fed. Cir. 1999) ("The possibility of [the claimed] alignment does not legally suffice to show anticipation . . .. Occasional results are not inherent."); *Bettcher Ind., Inc. v. Bunzi USA, Inc*., 661 F.3d 629, 640 (Fed. Cir. 2011) ("The speculative notion that by happenstance the [prior art] chamfers might, under hypothetical circumstances, be capable of operating as a bearing race [as claimed] is an insufficient basis to mandate overturning the jury's verdict [of no anticipation].")

In summary, Synopsys in its Appeal Brief conjures up for the first time particular test conditions of a functional simulation of the Gregory circuit illustrated in Fig. 9. These test conditions were fabricated by Synopsys, are not

disclosed (either explicitly or inherently) in Gregory, are entirely unsupported by any evidence in the record, and stand in contrast to the substantial evidence provided by Mentor in the form of Dr. Sarrafzadeh's opinions.

2) Synopsys' new theory regarding Figures 16 and 18 of Gregory

Synopsys also presents an entirely new theory of anticipation based on figures 16 and 18 of Gregory. Br. 54-58. Figures 16 and 18 were cited in Synopsys' Petition for disclosing the features of claim 28, but no explanation was provided of how the figures might disclose "execution status" other than to refer to the analysis of Fig. 9 with respect to claim 1. A180.[5]

Mentor having shown that Synopsys' analysis of claim 1 (and claim 28) is deficient, Synopsys now attempts to offer a particular simulation scenario of the circuit illustrated in Fig. 18 of Gregory to show that the Fig. 18 "tempout" signals indicate execution status. This new Fig. 16/18 analysis fails for the same reason the new Fig. 8/9 analysis fails. *See supra* Argument, § III.B.1). The functional simulation example is not explicitly disclosed in Gregory, and it is also not inherent in Gregory, because: 1) functional simulation of Fig. 18 is not necessarily present in the processes disclosed in Gregory, 2) even if the Fig. 18 circuit was functionally simulated, the precise manner suggested by Synopsys for functionally

[5] To be clear, Synopsys did not assert figures 16 and 18 as anticipatory of claim 1 in the Petition.

simulating the Fig. 18 circuit is not necessarily present in the processes disclosed in Gregory, and 3) many possible scenarios of functionally simulating the Fig. 18 circuit do not lead to the "tempout" signals indicating execution status of the related source code in Fig. 16.

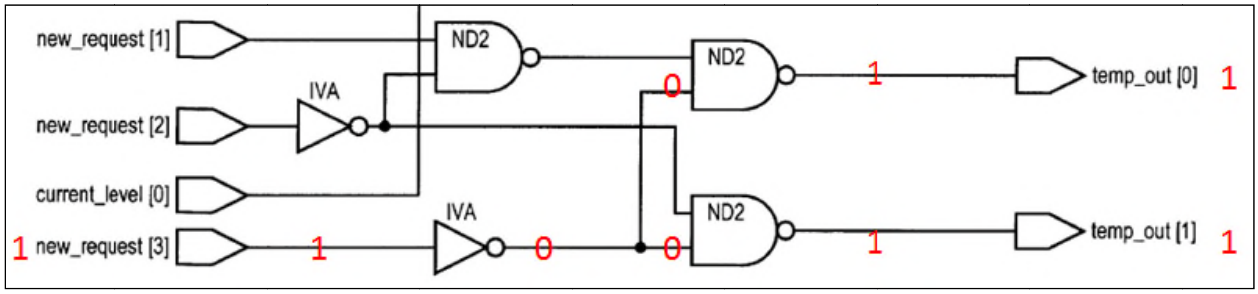
With respect to these last two points, had Synopsys presented its argument below, Mentor could have presented evidence supporting counter-examples that prove the example presented by Synopsys was merely a possibility, and not a certain result as required for inherent anticipation by Gregory. One such counter-example is outlined below to illustrate this point.

In Synopsys' description of the Fig. 18 input signals new_request(3:1)[6] and output signals tempout(1:0), two pieces of critical information have been omitted. Br. 54-58. Synopsys' first annotated copy of Fig. 18 (*Id.* at 56) is reproduced below.

[6] new_request(3:1) is shorthand for signals new_request(3), new_request(2), and new_request(1)

```
--Synopsys block_probe_begin
decode: process(new_request)
begin
  if(new_request(3) = '1') then
    new_level <= "11";
  elsif(new_request(2) = '1') then
    new_level <= "10";
  elsif(new_request(1) = '1') then
    new_level <= "01";
  else
    new_level <= "00";
  end if;
end process;
--Synopsys block_probe_end
```

Gregory, Fig. 16



Synopsys Annotation of Gregory, Fig. 18

Synopsys' first omission is that multiple possible input values of new_request(3:1) result in the same value output on tempout(1:0). This is because the four branches of the "if" statement in the HDL code in Fig. 16 are evaluated in succession. Specifically, when the "if" statement is performed, the first branch (if (new_request(3) – '1') . . .;) is tested first, and if true, its assignment statement (new_level < "11";) is then executed, and no subsequent statement is executed. A1996 (Ex. E to Sarrafzadeh Decl.) (definition of "If" statement in VHDL

Standard exhibit). If the first branch tests false, then the second branch (elsif(new_request(2) = '1') is tested, and so on. As a result, if new_request(3) is a "1", than the values of new_request(2) and new_request(1) do not matter, i.e., input values of "100", "101", "110", or "111" all result in the same output value "11".[7]

Synopsys' second omission is that the "if" statement in Fig. 16 is contained within a "process" statement identified at the beginning by "decode: process (new_request)" and at the end by "end process." The argument inside the parenthesis, i.e., "(new_request)," is referred to as a sensitivity list, and it indicates that the contents of the process, i.e., the "if" statement, is executed only when there is a change in values of at least one of the signals in the sensitivity list. In this case, the sensitivity list includes the input signals new_request(3:1),[8] and thus, the process including the "if" statement executes only when the value of at least one of these signals changes.

With these two facts in mind: 1) multiple different input combinations can lead to the same output result, and 2) the "if" statement executes in response to a

[7] This can be seen in the first annotated copy of Fig. 18 presented by Synopsys on page 56 of its Appeal Brief in that the values of new_request(2) and new_request(1) need not be propagated through the logic for a "11" to be determined at the output.

[8] Identified in Fig. 16 simply as "new_request."

change in the input combination; it is seen that a change in the input combination could cause an execution of one of the branches of the "if" statement that may not be reflected in the output signals tempout(1:0). For example, in response to a change of the new_request(3:1) from "100" to "101," the output signals tempout(1:0) would stay static from a previous value "11" to a new value "11". Because the tempout signals would not change, an observer of these signals would have no indication that the "if" statement executed.

Like Synopsys' new Fig. 8/9 theory discussed above, Synopsys errs in presuming that: (a) the circuit in Fig. 18 would be simulated, and (b) the Fig. 18 circuit would be simulated in a particular way. Neither of these presumptions are explicitly or inherently disclosed in Gregory. *C.f., In re Oelrich*, 666 F. 2d at 581; *Milgraum,* 192 F.3d at 1365; *Bettcher*, 661 F.3d at 640 (Fed. Cir. 2011).

IV. THE BOARD IS NOT REQUIRED TO ADDRESS PATENT CLAIMS THAT ARE OUTSIDE THE SCOPE OF THE *INTER PARTES* REVIEW

The Court should reject Synopsys' argument (at 68-76) that the Board erred by failing to address in its final written decision the patentability of claims with respect to which the Board did not institute *inter partes* review. This argument is not properly before the Court and lacks merit.

A. Synopsys Waived Its Argument Regarding The Scope Of The Inter Partes Review

"Simple fairness to those who are engaged in the tasks of administration, and to litigants, requires as a general rule that courts should not topple over administrative decisions unless the administrative body not only has erred but has erred against objection made at the time appropriate under its practice." *United States v. L.A. Tucker Truck Lines*, 344 U.S. 33, 37 (1952). Accordingly, "[i]t is well-established that a party generally may not challenge an agency decision on a basis that was not presented to the agency." *In re DBC*, 545 F.3d 1373, 1378 (Fed. Cir. 2008).

Synopsys did not argue to the Board that it was required to review not only the claims with respect to which the Board instituted review (claims 1-9, 11, and 28-29) but also the claims with respect to which the Board declined to institute review (claims 10, 12-15, 20-27, and 30-33). Synopsys could have raised a challenge to the scope of the *inter partes* review by requesting partial rehearing of the Board's institution decision. *See* 37 C.F.R. § 42.71(c), (d); *see also DBC*, 545 F.3d at 1379 (finding waiver when a party could have raised its argument before the agency "in a post-argument submission or in a motion for reconsideration"). Synopsys also could have raised the issue by motion. *See* 37 C.F.R. § 42.20. And Synopsys could have raised the issue in its reply brief during the trial. It did none of these things. Instead, Synopsys' brief "request[ed] that the Board invalidate

claims 1-9, 11, 28, and 29," A567, and did not mention claims 10, 12-15, 20-27, and 30-33 in its discussion of "the challenged claims," *see* A560-567.

By failing to argue before the Board that it lacked discretion to institute review with respect to only some of the claims raised in the petition, and by failing to ask the Board to issue a final written decision addressing all of the claims identified in the petition, Synopsys has waived (or forfeited) the issue. The Court should reject Synopsys' argument on that basis alone.

B. The Board Has Discretion To Limit Its Final Written Decision To Patent Claims On Which The Petitioner Demonstrates A Reasonably Likelihood Of Success

Synopsys' argument also fails on the merits. The PTO has construed the AIA not to require the Board to waste scarce resources on claims when the petitioner has not demonstrated a reasonable likelihood of success. The statute's text and purpose support the PTO's approach.

As a general matter, agencies may adopt procedures for summary decisionmaking in cases where the party requesting agency action fails to satisfy a threshold requirement for a full hearing. *See, e.g.*, *Federal Power Comm'n v. Texaco, Inc.*, 377 U.S. 33, 39-45 (1964) ("[T]he statutory requirement for a hearing . . . does not preclude the Commission from particularizing statutory standards through the rulemaking process and barring at the threshold those who [fail to] measure up to them . . .."); *United States v. Storer Broadcasting Co.*, 351

U.S. 192, 205 (1956) ("We do not think Congress intended the Commission to waste time on applications that do not state a valid basis for a hearing.").

Congress enacted the AIA against this doctrinal backdrop. Indeed, it specifically directed the PTO, in prescribing regulations governing *inter partes* review, to consider "the efficient administration of the Office, and the ability of the Office to timely complete proceedings instituted under this chapter." 35 U.S.C. § 316(b). The PTO acted well within its authority in promulgating a regulation that limits *inter partes* review proceedings to claims with respect to which the petitioner demonstrates a reasonable likelihood of success. Under that regulation, "[w]hen instituting *inter partes* review, the Board may authorize the review to proceed on all or some of the challenged claims and on all or some of the grounds of unpatentability asserted for each claim." 37 C.F.R. § 42.108(a). "Any claim or issue not included in the authorization for review is not part of the review." *Changes to Implement Inter Partes Review Proceedings, Post-Grant Review Proceedings, and Transitional Program for Covered Business Methods*, 77 Fed. Reg. 48,680, 48,689 (Aug. 14, 2012). In its rulemaking, the PTO explained:

> By limiting the review in such a manner, the patent owner is provided with a defined set of potentially meritorious challenges and will not be burdened with responding to non-meritorious grounds that fail to meet the initial thresholds. This convergence of issues for review streamlines the proceeding and aids in the efficient operation of the Office and the ability of the Office to complete the proceeding within the one-year

> timeframe. It is inefficient and unfair to patent owner to require a full response to challenges on claims that do not meet the initial threshold.

77 Fed. Reg. at 48,703.

Contrary to Synopsys' contention (Br. 68-76 & n.18), the AIA does not prohibit the PTO's approach. Synopsys largely ignores § 314, the provision of the AIA that governs the institution of *inter partes* review. While § 314 provides that the Board "*may not* authorize an *inter partes* review to be instituted *unless*" specific conditions are met, 35 U.S.C. § 314(a) (emphasis added), it does not require the Board to institute review under any circumstances, and it is silent about whether the Board has discretion to limit an *inter partes* review to a subset of the claims challenged in the petition.

Similarly, nothing in § 316, which governs the conduct of *inter partes* reviews, supports Synopsys' position. Instead, § 316 makes clear that the Board often will have no reason to discuss in its final written decision all of the claims attacked in the petition for review, either because the patent holder has "[c]ancel[ed] any challenged patent claim" or "propose[d] a reasonable number of substitute claims." 35 U.S.C. § 316(d). Congress surely did not intend § 318(a) to require the Board to opine on the patentability of claims that the patent holder abandons during the *inter partes* review. Synopsys' reading of § 318(a) to require

a decision with respect to every claim challenged in a petition would require just that absurd result.

Synopsys argues that § 318(a)—read in isolation—requires the Board's final written decisions to address every patent claim identified in every petition that it grants. Synopsys misreads § 318(a), which by its plain terms addresses decisions of the Board rather than the institution or scope of *inter partes* review proceedings. Section 318 provides: "If an inter partes review is instituted and not dismissed under this chapter, the [Board] shall issue a final written decision with respect to the patentability of any patent claim challenged by the petitioner and any new claim added under section 316(d)." 35 U.S.C. § 318(a).

Synopsys argues that the phrase "any patent claim challenged by the petitioner" can only mean any claim mentioned in the petition for *inter partes* review. But read in context, the phrase is correctly interpreted to mean any claim that the petitioner challenges at the time of the final written decision. The Board's final written decision need not address the patentability of claims with respect to which the petition offers no meritorious basis for finding invalidity. If Congress had intended to require the Board to institute and conduct *inter partes* review with respect to all of the claims in a petition or none of them, it would have said so expressly in § 314 or § 316 instead of implicitly in § 318.

In construing § 318(a), moreover, courts must "look not only to the particular statutory language, but to the design of the statute as a whole and to its object and policy." *Pitsker v. Office of Personnel Mgmt.*, 234 F.3d 1378, 1381 (Fed. Cir. 2000) (quoting *Crandon v. United States*, 494 U.S. 152, 158 (1990)). Read in context, § 318(a) requires only that the Board address the patentability of claims that remain at issue at the time of the final written decision.

Practical considerations also weigh against Synopsys' "all or nothing" approach in which the Board must grant a petition for *inter partes* review in full or not at all. Congress expects the Board to complete *inter partes* review proceedings on an expedited basis. The AIA generally requires the Board to decide whether to institute *inter partes* review within three months after the preliminary response to the petition is filed, 35 U.S.C. § 314(b), and to issue a final written decision within one year after the institution decision, *id.* § 316(a)(11) . Indeed, Congress was so concerned about ensuring the timeliness of *inter partes* review that it preferred to have the Board reject even meritorious petitions that would threaten the agency's timely completion of pending proceedings. *See* 157 Cong. Rec. S1377 (daily ed. Mar. 8, 2011) (statement of Sen. Kyl) (explaining that the AIA "reflects a legislative judgment that it is better that the Office turn away some petitions that otherwise satisfy the threshold for instituting an *inter partes* or post-grant review than it is to allow the Office to develop a backlog of instituted reviews that

precludes the Office from timely completing all proceedings."); *cf.* 77 Fed. Reg. at 48,703 (explaining that the PTO's approach "streamlines the proceeding and aids in the efficient operation of the Office and the ability of the Office to complete the proceeding within the one-year timeframe").

Synopsys and *amicus curiae* the SAS Institute note that the institution of review with respect to only some claims identified in a petition narrows the preclusive effect of the Board's *inter partes* review. *See* Br. 74; SAS Br. 5-7. While it is true that petitioners will not be estopped in civil actions from challenging the validity of claims with respect to which the Board does not issue a final written decision, 35 U.S.C. § 315(e)(2), district courts may look to these decisions for guidance and should not be unduly burdened in deciding the validity of claims with respect to which the petitioner offered no meritorious argument before the Board.

Whether the PTO's current approach or the one Synopsys advocates would leave more patent invalidity determinations to district courts is an empirical question with no clear answer. But neither Synopsys nor the SAS Institute offers any basis for concluding that Congress intended the answer to this question to trump its clear concerns about the timeliness of the Board's decisions. That Synopsys' approach would increase the time that the Board must devote to each *inter partes* review counsels against its interpretation of the statute.

Nor is there merit to Synopsys' assertion that Congress required the Board's final written decisions to address all of claims identified in a petition so as to enable judicial review with respect to those claims. Synopsys offers no reason to conclude that Congress intended this Court to review the Board's finding that a petitioner has no reasonable likelihood of prevailing on a claim when (but only when) the Board grants the petition with respect to other claims. And Synopsys is not in a position to make this argument in any case due to its failure to ask the Board to address in its final written decision all of the claims identified in the petition.

In sum, the AIA requires the Board to issue a final written decision with respect to the patentability of any claim in dispute at the conclusion of the proceeding. The Board did not err by failing to address claims that Synopsys listed in its petition but for which it demonstrated no reasonable likelihood of prevailing.

V. THE BOARD SHOULD HAVE DISMISSED THE IPR UNDER 35 U.S.C. § 315(b)

The AIA provides that "[a]n inter partes review may not be instituted if the petition requesting the proceeding is filed more than 1 year after the date on which the petitioner, real party in interest, or privy of the petitioner is served with a complaint alleging infringement of the patent." 35 U.S.C. § 315(b). That

provision barred the IPR in this case, because EVE is both a privy of Synopsys and a real party in interest.

EVE and Synopsys sought to avoid the § 315(b) bar through a carefully choreographed series of actions.

- On September 26, 2012, Synopsys filed a petition for *inter partes* review of the patent. A135-199.
- At 8:30 a.m. on September 27, 2012, Synopsys and EVE filed a joint action for a declaratory judgment of invalidity and non-infringement concerning the '376 patent. A1583, A1573-1588. The Complaint alleged that Synopsys would in the immediate future be "using, importing, selling, offering for sale and/or supporting the ZeBu Products in the United States, which line of products was previously accused by [Mentor] of infringing the . . . '376 patent[]." A1577.
- The Complaint alleged that Synopsys had "entered into an agreement to acquire the business of EVE" earlier on September 27. A1577.
- No later than October 4, 2012, EVE became a wholly-owned subsidiary of Synopsys. A1590, A1592.

The PTO acknowledges that Synopsys and EVE were in privity by October 4 -- well before the Board instituted the *inter partes* review. *See* Rea Mem. at 24, *Mentor Graphics Corp. v. Rea*, No. 1:13-cv-518 (E.D. Va.) (ECF No. 38) (filed

May 24, 2013) (hereinafter "PTO D. Ct. Mem.") ("[A]t least as of October 4, 2012, Synopsys and EVE are in privity."). The Board erred, however, in concluding that Synopsys and EVE could circumvent § 315(b)'s privity bar by choreographing their transaction so that Synopsys did not formally acquire EVE until just after Synopsys filed its petition. Section 315(b) does not permit such gamesmanship.

The Board also erred in holding that EVE is not a real party in interest. Synopsys filed the petition in anticipation of its acquisition of EVE; EVE was a primary beneficiary of the *inter partes* review; and the two companies tightly coordinated the nearly-simultaneous petition, acquisition, and joint declaratory judgment action against Mentor.

Because the petition was filed more than one year after EVE, a privy of the petitioner and a real party in interest, was served with a complaint alleging infringement, the Board should have dismissed the *inter partes* review.

A. The AIA Does Not Preclude All Judicial Review Of The Privity And Real Party In Interest Issues

Courts apply a "strong presumption" that Congress does not intend to preclude judicial review of agency action. *See, e.g.*, *Bowen v. Mich. Acad. of Family Physicians*, 476 U.S. 667, 672 (1986).). Judicial review ensures that agencies respect the limits of their authority, adhere to laws enacted by Congress, and treat parties fairly, consistently, and rationally. *See, e.g., Motor Vehicle Mfgs. Ass'n of the United States Inc. v. State Farm Mutual Auto. Ins. Co.*, 463 U.S. 29,

43-44 (1983); *Adkins v. United States*, 68 F.3d 1317, 1323 (Fed. Cir. 1995). For these reasons, courts will not interpret a statute to preclude all judicial review absent "clear and convincing evidence of a contrary legislative intent." *Abbott Labs v. Gardner*, 387 U.S. 136, 141 (1967) (quotation marks omitted); *Block v. Cmty Nutrition Inst.*, 467 U.S. 340, 350-51 (1984). Courts may, however, determine that Congress has *channeled* judicial review to a particular court at a particular time as part of a complex regulatory regime. *See Thunder Basin Coal Co. v. Reich*, 510 U.S. 200, 207 (1994); *Block*, 467 U.S. at 351.

When Mentor challenged the Board's privity determination in district court, the PTO contended that the Board's determination *is* subject to judicial review, but only in this Court and only after the Board's FWD. *See, e.g.*, PTO D. Ct. Mem. at 2. More recently, the PTO has argued that that there can be *no* judicial review by any court, at any time, of the Board's application of § 315. *See* PTO Br. at 30, *In re Cuozzo Speed Techs., LLC*, No. 2014-1301 (Fed. Cir.) (filed June 5, 2014) [hereinafter "PTO *Cuozzo* Br."]. The PTO had it right the first time. Congress did not override the strong presumption in favor of judicial review.

1) The AIA Allows The Court To Review The Board's § 315(b) Analysis As Part Of Its Review Of The Board's FWD

Nothing in the AIA's text prohibits all judicial review of the Board's application of § 315(b). The PTO points to § 314(d), which provides that "[t]he determination by the Director whether to institute an inter partes review under this

section shall be final and nonappealable." 35 U.S.C. § 314(d). This Court has read § 314(d) to prohibit "immediate review of a decision to institute an *inter partes* review," *In re Proctor & Gamble Co.*, 749 F.3d 1376, 1379 (Fed. Cir. 2014), including when the patent holder challenges the Board's authority under § 315(b), *id.* at 1378 & n.1. The Court recognized, however, that "[i]t is a separate question whether section 314(d) means that the decision to institute the review is unchallengeable later—if the Board reaches a decision under section 318(a) and an appeal is taken under section 319." *Id.* at 1379.[9]

The correct answer to that "separate question" is "no." Section 319 expressly authorizes a "party dissatisfied with the final written decision of the [Board] under section 318(a)" to "appeal the decision pursuant to sections 141 through 144." 35 U.S.C. § 319. As the PTO has explained:

> Nothing in the statutory scheme limits the reasons that a party might be so 'dissatisfied,' and this could include the argument that the [Board] lacked the authority to issue a written determination at all. And thus, nothing in the statutory scheme precludes Mentor from presenting

[9] The Court has also held that it lacks authority to review the Board's decision *not* to institute *inter partes* review. *See St. Jude Med. Cardiology Div., Inc. v. Volcano Corp.*, 749 F.3d 1373, 1375 (Fed. Cir. 2014)) (appeal); *In re Dominion Dealer Solutions*, 749 F.3d 1379, 1381 (Fed. Cir. 2014). These decisions are distinguishable on the same ground as *Proctor & Gamble*, and on the additional ground that discretionary agency decisions not to institute administrative proceedings generally are non-reviewable. *See Heckler v. Chaney*, 470 U.S. 821, 831 (1985).).

> this issue to the Federal Circuit from a *potential* adverse written determination on the '376 patent by the [Board].

PTO D. Ct. Mem. at 13; *see also id.* at 18 ("[T]he statutory scheme does not preclude judicial review . . . at the Federal Circuit."). Thus, a challenge to the Board's application of § 315(b) in this case is properly viewed as seeking judicial review of the Board's *final* decision to invalidate patent claims on the ground that the final decision was *ultra vires*; it is not a challenge to the Board's decision to institute the inter *partes* review. This interpretation is consistent with the general administrative law principle that "[a] preliminary, procedural, or intermediate agency action or ruling not directly reviewable is subject to review on the review of the final agency action." 35 U.S.C. § 704.

To be sure, § 319 authorizes aggrieved parties to appeal "the final written decision of the [Board] under section 318(a)," and § 318(a) directs the Board to "issue a final decision with respect to the patentability of any patent claim," 35 U.S.C. §§ 318, 319 (emphases added). But that statutory language does not demonstrate Congress's clear intent to preclude judicial review. Indeed, it does not address judicial review at all. Instead, Section 318(a) identifies issues that a FWD "shall" address without prohibiting the Board from addressing other issues – as the Board did in this very case by addressing the § 315(b) issues at length in its FWD. A11-17; *see also* PTO D. Ct. Mem. at 22 ("To be sure, no rule or regulation *requires* the PTAB to revisit its 315(b) decision, but similarly no rule or regulation

prevents it from doing so either."). Even if § 318(a) limited judicial review to issues "with respect to the patentability of any patent claim," the question whether the Board had authority to adjudicate patentability fits within that broad category.

In sum, the statutory language does not overcome the strong presumption in favor of judicial review. The Board is not the sole arbiter of its own authority, exempt from any supervision by this Court or the Supreme Court. The fact that the PTO initially argued strongly in *favor* of this result confirms that the statutory language does not clearly foreclose it.

Practical considerations strongly reinforce this conclusion. Without judicial review, there is be no way to prevent the Board from exceeding its statutory authority. For example, the Board would be acting unlawfully if it instituted an *inter partes* review three years after the patent owner served the petitioner with a complaint alleging infringement of the challenged patent. *See* 35 U.S.C. § 315(b). Under the PTO's new interpretation, however, this Court would be powerless to do anything about it. Instead, the Court would be required to review the patentability of the claims at issue -- even if the entire administrative proceeding was unauthorized and *ultra vires*.[10] Congress could not have intended to place the Court in this untenable position.

[10] This case illustrates the need for judicial review. The Board initially determined that the privity bar applies only if the petitioner has an ownership interest in the

This Court's review of the Board's application of § 315(b) after an *inter partes* review has concluded does not implicate any of the PTO's concerns regarding immediate review of the Board's institution decisions. It avoids the inefficiency of "piecemeal review," PTO D. Ct. Mem. at 17, and will not hinder the Board as it seeks to complete *inter partes* reviews within one year of institution, *id.* at 17-18. Moreover, the "presumption of judicial review is a repudiation of the principle that efficiency of regulation conquers all." *Sackett v. EPA*, 132 S. Ct. 1367, 1374 (2012).

2) Judicial Review Is Available For The Additional Reason That The Board's FWD Expressly Addresses The Privity Issue

Even if the AIA generally precluded any judicial review of the Board's application of § 315(b), this appeal would present an exception to that general rule. The Board's final decision addresses the § 315(b) issue at length. A52-57. As noted above, a party who is dissatisfied with the Board's FWD is authorized to appeal it to this Court. *See* 35 U.S.C. § 319. Because Mentor is dissatisfied with the Board's application of § 315(b) in its FWD, it is authorized to seek review pursuant to the express language of § 319.

---

patent. The Board abandoned that clearly incorrect position only after Mentor sought judicial review.

B. Synopsys And EVE Are Privies For Purposes Of § 315

As the PTO explained in the district court, "[e]veryone – including the PTAB – agrees that at least as of October 4, 2012, Synopsys and EVE are in privity." PTO D. Ct. Mem. at 24. The issue is whether Synopsys and EVE were privies "at the relevant points in time." *Id.* The plain language of § 315(b) makes clear that the only "relevant time[]" is the date of the Board's decision to institute the *inter partes* review.

1) The Acquisition Of EVE By Synopsys Made EVE A Privy Of Synopsys

The PTO defines the term "privy" in § 315(b) in accordance with its common law meaning. OPTPG, 77 Fed. Reg. at 48,759-60. Under the common law, when one company acquires the entire business of another company, the new parent and subsidiary are generally are treated as privies. *See, e.g., Golden State Bottling Co. v. NLRB*, 414 U.S. 168, 177-80 (1973); *Doe v. Urohealth Sys., Inc.*, 216 F.3d 157, 162 (1st Cir. 2000); *Lubrizol Corp. v. Exxon Corp.*, 929 F.2d 960, 966 (3d Cir. 1991); *In re Teltronics Servs., Inc.*, 762 F.2d 185, 191-92 (2d Cir. 1985); *Pan Am. Match Inc. v. Sears, Roebuck & Co.*, 454 F.2d 871, 874 (1st Cir. 1972).

2) Section 315(b) Applies Because EVE And Synopsys Were In Privity Before The Board Instituted Inter Partes Review

Under § 315(b), the acquisition of EVE by Synopsys before the Board's decision to institute the *inter partes* review deprived the Board of authority to conduct the proceeding.

Section 315(b) provides that "[a]n inter partes review *may not be instituted* if . . . the petitioner, real party in interest, or privy of the petitioner" is served with a complaint more than one year before the petition is filed. 35 U.S.C. § 315(b) (emphasis added). An *inter partes* review is "instituted" by a decision of the Board, not by the filing of a petition or any other event. *See* A1. Because the text of § 315(b) speaks to whether the Board may institute review, the relevant event for determining the status of privies is the Board's decision to institute review. *Cf. Republic of Austria v. Altmann*, 541 U.S. 677, 697-98 (2004) (looking to the statutory language to identify the "relevant conduct regulated by the Act").

Practical considerations reinforce the statutory language. If the choreographed maneuvers that Synopsys and EVE employed in this case pass judicial muster, they will serve as a blueprint for other entities seeking to negate the one-year bar of § 315(b). Congress could not have intended to permit this kind of gamesmanship.

Thus, both the text and purpose of § 315(b) confirm that the Board's patentability decision was *ultra vires*.

3) The Board's Arguments To the Contrary Are Unpersuasive

Initially, the Board determined that Synopsys' acquisition of EVE was "irrelevant" because "[t]he only property right at issue in this proceeding is that of the '376 patent," not "the property interest in EVE's products." A17 (citing *Int'l Nutrition Co. v. Horphag Research, Ltd.*, 220 F.3d 1325, 1329 (Fed. Cir. 2000)). This position was clearly incorrect. It violated the settled rule that successors in interest are in privity with their predecessors, and it effectively repealed the privity limitation in § 315(b) (since accused infringers rarely if ever own the patents they are accused of infringing). The Board did not defend this position in district court, *see* PTO D. Ct. Mem. at 24-30, and dropped it from its FWD, *see* A52-55. Although the Board has shifted to other arguments for refusing to apply the privity bar, those arguments also fail to withstand scrutiny.

(a) Section 315(b) is not limited to privity relationships established before the filing of a petition for *inter partes* review

The Board did not rely on the language of the statute when it concluded that it could proceed with an *inter partes* review because Synopsys and EVE were not in privity at the moment Synopsys filed its petition for *inter partes* review. *See* A52-55. Instead, the Board cited a regulation, 37 C.F.R. § 42.101(b), which it said "makes clear that it is only privity relationships up until the time a petition is filed that matter." A53.

The Board's reliance on 37 C.F.R. § 42.101(b) is misplaced. The regulation has nothing to do with the relevant date for determining privity. In relevant part, it reads:

> **§42.101 Who may petition for *inter partes* review**
>
> A person who is not the owner of a patent may file with the Office a petition to institute an inter partes review of the patent unless . . .
>
> (b) The petition requesting the proceeding is filed more than one year after the date on which the petitioner, the petitioner's real party-in-interest, or a privy of the petitioner is served with a complaint alleging infringement of the patent.

37 C.F.R. § 42.101. As its title makes clear, the regulation identifies "[w]ho may petition for *inter partes* review," *id.*; it does not specify a cut-off date for the § 315(b) analysis and is silent about the relevance of events between the filing of a petition and the Board's institution decision.

Moreover, the Board's argument is at odds with the PTO's decision to reject bright-line rules to implement § 315(b) in favor of a "case-by-case approach . . . based on controlling case law and the particular facts of each case." 77 Fed. Reg. at 48,694. In accordance with this decision, the PTO's regulation simply "follows the statutory language of 35 U.S.C. § 315(b)." *Id.* at 48,688.[11]

[11] Because 37 C.F.R. § 42.101 simply "follows the statutory language," *id.*, the Board's interpretation of the regulation is not entitled to deference under *Auer v. Robbins*, 519 U.S. 452 (1997).). *See Gonzales v. Oregon*, 546 U.S. 243, 257 (2006).). Nor is the Board's interpretation of § 315(b)) entitled to deference under

Even if § 42.101(b) meant what the Board now suggests, it would be invalid. As discussed above, *see supra* Argument, V.B.2), § 315(b) itself specifies that the date of the institution decision is the cut-off date for evaluating whether the petitioner and a non-party are in privity. If Congress had intended events between the filing of a petition and the institution of review to be irrelevant, it would have written the statute differently. *See Util. Air Regulatory Group v. EPA*, 134 S. Ct. 2427, 2446 (2014) ("[A]n agency may not rewrite clear statutory terms to suit its own sense of how the statute should operate.").

(b) Section 315(b) is not limited to privity relationships established before or during the patent holder's infringement action

In its decision to institute the *inter partes* review, the Board suggested that § 315(b) is inapplicable because Synopsys and EVE were not in privity in 2006 when Mentor sued EVE for infringement. *See* A16. The Board subsequently denied Mentor's motion for discovery on the ground that "[t]he legal standard adopted by the Board is that § 315(b) requires a privity relationship 'in 2006 when EVE was served with a complaint alleging infringement of the '376 patent,'"

Chevron, U.S.A., Inc. v. Natural Res. Def. Council, Inc. 467 U.S. 837 (1984).). "The Board's statutory interpretation in a particular case is given no deference." *In re Swanson*, 540 F.3d 1368, 1374 n.3 (Fed. Cir. 2008).). Moreover, the Board's interpretation of § 315(b)) is "contrary to clear congressional intent," *Chevron*, 467 U.S.. at 843 n.9, and the agency purports to apply the common law doctrine of privity instead of exercising any administrative expertise.

A380. The Board's FWD de-emphasizes this argument, referring to it only as "another reason to conclude that there was no privity relationship between Synopsys and EVE sufficient to trigger § 315(b)'s prohibitions." A54.

To the extent the Board continues to regard 2006 as the cut-off date for determining privity, its position lacks merit. At common law, "[s]uccessive property relationships provide some of the oldest and best established rules for extending preclusion to nonparties." 18A Charles Alan Wright & Arthur R. Miller, Federal Practice and Procedure § 4462, at 658 (2d ed. 2002). Indeed, legislators expressly discussed successor-in-interest privity while considering section 315(b). *See, e.g.*, 154 Cong. Rec. S9987 (daily ed. Sept. 27, 2008) (statement of Sen. Kyl) ("One situation in which parties have frequently been held to be in privity is when they hold successive interests in the same property."); 157 Cong. Rec. S5432 (daily ed. Sept. 8, 2011) (statement of Sen. Schumer) ("[a] 'privy' is a party that has a direct relationship to the petitioner with respect to the allegedly infringing product or service. . .."). There is no reason to conclude that Congress intended to eliminate this core aspect of common law privity doctrine.

(c) Section 315(b) applies regardless of whether the infringement action has ended

Finally, the Board suggested that "this case does not implicate the concerns that [§ 315(b)] appears designed to address" because Synopsys did not file the petition to "inject delay into an already-pending litigation." A54. This argument

fails because the text of Section 315(b) broadly prohibits institution of *inter partes* review if the petition "is filed more than 1 year after the date on which the petitioner, real party in interest, or privy of the petitioner is served with a complaint alleging infringement of the patent." 35 U.S.C. § 315(b). Nothing in the text suggests § 315(b) becomes inoperative when the litigation between the patent owner and the petitioner (or its privy) ends. "If the language of the statute is clear and its meaning unambiguous, that is the end of [the] inquiry." *Electrolux Holdings, Inc. v. United States*, 491 F.3d 1327, 1330 (Fed. Cir. 2007) (citations omitted).

"In arguing that one of the purposes of the [AIA] trumps the statutory text, the [Board] improperly elevates an interpretive tool to a command." *Bartels Trust for Benefit of Cornell Univ. v. United States*, 617 F.3d 1357, 1361 (Fed. Cir. 2010). In addition, the Board focuses on one purpose of § 315(b)—preventing petitioners from "inject[ing] delay into an already pending litigation," A54—while ignoring another important purpose—protecting patent owners from having to defend before the Board patents they have already defended successfully in court. *See* OPTPG, 77 Fed. Reg. at 48,759 (§ 315(b) "seeks . . . to prevent parties from having a 'second bite at the apple.'").[12]

[12] The committee report and floor statement cited by the Board note that the pending bills would impose time limits on petitioning for *inter partes* review once

C. EVE Is Also A Real Party In Interest

The Board exceeded its authority for an additional reason: EVE was a real party in interest to the *inter partes* review proceedings. The Act does not define "real party in interest," but courts and the PTO alike recognize that determining whether a nonparty is a real party in interest requires a "flexible" analysis focused "'on the particular facts and circumstances' of the case." *Huff v. Comm'r*, 743 F.3d 790, 796 (11th Cir. 2014) (quoting *Chiles v. Thornburgh*, 865 F.2d 1197, 1214 (11th Cir. 1989)); *see* OPTPG, 77 Fed. Reg. at 48,759.

The OPTPG provides that a nonparty may be a real party in interest if it "exercised or could have exercised control over a party's participation in the proceeding." *Ibid.* But any substantial relationship between the nonparty and the proceeding is generally sufficient. *See* OPTPG, 77 Fed. Reg. at 44,759–44,760. "Relevant factors include" the nonparty's "relationship with the petitioner," its "relationship to the petition itself, including the nature and/or degree of involvement in the filing," and "the nature of the entity filing the petition." *Id.* at 44,760.

---

a lawsuit is filed, but are silent about how § 315(b)) applies once the lawsuit has ended. *See* A54-55 (citing H.R. Rep. No. 112-98, at 45 (2011), and 157 Cong. Rec. S1326 (daily ed. Mar. 7, 2011)) (statement of Sen. Sessions)). Other statements indicate that § 315(b)) applies after the conclusion of an underlying infringement action. *See, e.g.*, 154 Cong. Rec. S9987 (daily ed. Sept. 27, 2008)) (statement of Sen. Kyl) ("The concept refers to a relationship between the party to be estopped and *the unsuccessful party in the prior litigation*.") (emphasis added)).

Application of those factors yields the conclusion that EVE was a real party in interest to the *inter partes* review proceedings. EVE clearly benefited from the petition for *inter partes* review, which Synopsys filed just before it consummated its acquisition of EVE and EVE's infringing ZeBu products. A412–417; *see* Black's Law Dictionary (9th ed. 2009) ("real party in interest" "generally . . . benefits from the action's final outcome"). Just as clearly, Synopsys filed the petition in anticipation of its acquisition of EVE, A412–417, thereby establishing a close "relationship with the petitioner," OPTPG, 77 Fed. Reg. at 48,760. Finally, EVE had a significant "relationship to the petition itself," *ibid.*, because Synopsys and EVE coordinated the acquisition, the petition, and the filing of their joint declaratory judgment action against Mentor, and likely collaborated regarding the content of the petition as well, *see* A412-417.

The Board concluded that Mentor had not shown "that Synopsys allowed EVE to direct or control content of the petition filed in this case." A56. That rationale is inadequate because it focuses on a single factor—direction and control—to the exclusion of all others. The Board also determined that Mentor had not adduced "other evidence that EVE was a real part-in-interest prior to the filing of the petition." A56. That determination improperly focuses on the date Synopsys filed the petition rather than the date the Board instituted the review, *see supra* Argument, V.B.2), and in any event, disregards record evidence indicating

that Synopsys and EVE had formed a close relationship prior to Synopsys' filing of the petition.

The Board also abused its discretion in denying Mentor's motion for additional discovery on the ground that, in the Board's view, any relationship between Synopsys and EVE that began after 2006 was simply irrelevant. A378-385. At a minimum, the Board should have authorized targeted discovery into the interactions between Synopsys and EVE around the time the petition was filed. The Board did not explain what more evidence Mentor needed to present in order to show that EVE was a real party in interest, A55-57, but it's denial of Mentor's discovery motion did deprive Mentor of the opportunity to develop that record. Particularly given the Board's obligation not to exceed its statutory authority, it was error for the Board to turn a blind eye to Synopsys and EVE's maneuvering.

VI. THE BOARD ERRED IN DENYING MENTOR GRAPHICS' MOTION TO AMEND

The Board also abused its discretion and acted arbitrarily and capriciously when it denied Mentor's contingent motion to substitute claims 35, 40, and 41 for claims 5, 8, and 9. Mentor introduced substitute independent claim 35 (A499-500) to explicitly recite language in accordance with the construction it urged for

the term "instrumentation signal."[13] A505. In denying Mentor's motion, the Board impermissibly shifted the burden of proving patentability of the proposed substitute claims from Synopsys to Mentor, contrary to 35 U.S.C. § 316(e). The Board also abused its discretion by unreasonably imposing on Mentor the need to show "patentability over the art in general." The imposition of this burden on the patent owner, coupled with imposed procedural constraints, effectively denied Mentor a reasonable opportunity to amend the claims.

A. The Board's Standard Erroneously Required Mentor to Carry the Burden of Showing Patentability

The Board found that Mentor's motion failed to satisfy the burden it imposed on Mentor to demonstrate patentability of the proposed amended claims over the prior art. A88 (FWD). This shift of the burden to prove patentability was contrary to 35 U.S.C. § 316(e).

Section 316 of the AIA authorizes patent owners to move to amend their patents in an IPR, 35 U.S.C. § 316(d), and provides that "the *petitioner* shall have the burden of proving a proposition of unpatentability by a preponderance of the evidence," *id.* § 316(e) (emphasis added). By its terms, § 316(e) does not limit the petitioner's burden of proving unpatentability to the claims of the patent as they

[13]Mentor argued for a narrower construction of "instrumentation signal" in original claim 5, which would have required the addition of logic to the gate level netlist. The Board declined to adopt this construction, and this is not challenged on appeal.

stood when the petition was filed. To the contrary, a natural reading of § 316(e) is that it extends to the entire proceeding, including amended claims proposed under § 316(d).

Reading § 316(e) to apply both to original claims and proposed substitutes is consistent with not only the statute's language but also the adjudicative nature of *inter partes* review proceedings. "Under an oppositional system[ ] the burden is *always* on the petitioner to show that a claim is not patentable." 154 Cong. Rec. S9987 (daily ed. Sept. 27, 2008) (statement of Sen. Kyl).

In placing the burden on Mentor, the Board did not acknowledge § 316(e) or the PTO's regulation addressing substitution of claims (37 C.F.R. § 42.121). The Board cited only 37 C.F.R. § 42.20(c), which contains only a general requirement that "the moving party . . . establish that it is entitled to the requested relief." A83. When applied in the context of determining the patentability of a proposed substitute claim, this standard is clearly contrary to the plain terms of § 316(e). In accordance with §316(e), it was legal error to shift the burden of proving unpatentability from Synopsys to Mentor with respect to the proposed substitute claims.

B. The Board's Standard Impermissibly Required Mentor To Show "General Patentability Over Prior Art" And Imposed Unreasonable Procedural Constraints

The Board correctly concluded that "Gregory does not show 'instrumentation logic comprising instrumentation logic circuitry that is additional to circuitry specified in the source code,' as required by proposed substitute claims 35, 40, and 41." A87. Regarding obviousness, Mentor explained in its Reply why "[t]he FPGA (emulation) related disclosures of Petitioner's cited Exhibits 1014-18 do not remedy Gregory's deficiencies." *See* A584-585. The Board did not refute Mentor's arguments, but rather merely stated in conclusory fashion its determination that Mentor had failed to meet its burden to show non-obviousness over Gregory. A88.

The Board further stated: "Moreover, distinguishing the proposed substitute claims only from the prior art references applied to the original claims is insufficient to demonstrate *general patentability over prior art*." *Id*. (emphasis added). This requirement that Mentor demonstrate general patentability over prior art was an abuse of discretion.

Mentor asserted that it believed Gregory was the closest art of record and that it was not aware of any prior art that would affect the patentability of the substitute claims. A509, A512-513. The Board determined that this was "insufficient, without discussing the level of ordinary skill in the art, and what was

previously known, with respect to each added feature, including the ordinary skill set possessed by such a hypothetical person." A88-89. According to the Board:

> In the context of the claim element added by Mentor Graphics, it is essential to know whether synthesizing source code, including additional instrumentation logic circuitry, pre-existed the claimed invention, in any context, and, if so, how it worked. Otherwise, Mentor Graphics is expected, reasonably to explain such pre-existing art, and why it would not have been applicable to render the invention of the proposed substitute claims obvious to one with ordinary skill in the art. Mentor has failed to do either.

A89.

But requiring Mentor to show "general patentability over prior art" in effect required proof of a negative—the nonexistence of invalidating prior art in *any* field. If not impossible to meet, the Board's requirement that the patent owner demonstrate "general patentability over prior art" is overly burdensome, impracticable, and unreasonable. For one thing, the standard provides no reasonable guideposts for ascertaining just how much prior art must be raised and patentably distinguished in order to satisfy the burden. Denying Mentor's motion because it did not cite to more *non-invalidating* prior art chides the patent owner for failing to fashion and knock-down straw arguments.

The Board also erred in coupling its unreasonable motion to amend standard with arbitrary and unreasonable procedural constraints, including its briefing page limitations. These constraints made it all the more impossible for Mentor to

comply with the onerous *Idle Free* standard.[14] It operated to effectively deny Mentor a full and fair opportunity to make its proposed amendments to the '376 patent.

In a ruling inconsistent with its rulings in other IPR trials,[15] the Board required Mentor to include its 4-page claims listing as part of its 15-page motion to amend, instead of attaching it as an appendix. A492. This ruling left Mentor with even less opportunity to establish "general patentability over prior art." The Board then denied Mentor's request (A589) for an increase in the 5-page limit for its reply, which Mentor sought in order to address the six new references Synopsys presented in an attempt to demonstrate the obviousness of the added features of substitute claim 35, taken in combination with Gregory. A538-541.

[14] Notably, there is a pervasive view in the patent community that the Board-imposed *Idle Free* standard and procedural constraints for motions to amend (*Idle Free v. Bergstrom*, IPR2012-0027, Paper 26 (June 11, 2013),), posted by the PTAB as "informational"), are unduly restrictive and in need of change. *See*, *e.g.*, American Bar Association – Intellectual Property Law Section (ABA-IPL) Comments on PTAB Trial Proceedings 7, available at http://www.uspto.gov/ip/boards/bpai/aba_ipl_20141016.pdf (October 16, 2014); and Intellectual Property Owners Association Comments on PTAB Trial Proceedings 4, *available at* http://www.uspto.gov/ip/boards/bpai/ipo_20140916.pdf (September 16, 2014)(responses to question No. 2).

[15] *See e.g.*, IPR2012-00006, Paper 33, April 18, 2013; IPR2013-00012, Paper 52, April 24, 2013; IPR2014-00441; Paper 19, Oct. 30, 2014 (claim listings in an appendix permitted).

By coupling an obligation to "demonstrat[e] general patentability over prior art" with inflexible procedural constraints, the Board denied Mentor a reasonable opportunity to show entitlement to its amended claims.

CONCLUSION

For the foregoing reasons, the Court should reverse the Board's FWD determination that this IPR is not barred under 35 U.S.C. § 315(b), and vacate the FWD on this basis. Alternatively, the Court should set aside the Board's decision, remand for further consideration of Mentor's motion to amend and any related proceedings. In any event, the Court should deny the relief sought by Synopsys.

Dated: December 17, 2014

Respectfully submitted,

/s/ Christopher L. McKee

Robert A. Long, Jr.
Matthew J. Berns
COVINGTON & BURLING LLP
One CityCenter
850 TENTH STREET, NW
WASHINGTON, DC 20001

Christopher L. McKee
Bradley C. Wright
Michael S. Cuviello
BANNER & WITCOFF, LTD.
Suite 1200
1100 13th Street, NW 20005
Telephone: (202) 824-3000

George A. Riley
Mark E. Miller
O'MELVENY & MYERS LLP
Two Embarcadero Center, 28th Floor
San Francisco, CA 94111
Telephone: (415) 984-8700

Attorneys for Cross-Appellant Mentor Graphics Corporation

ADDENDUM

Final Written Decision

Dated February 19, 2014

Trials@uspto.gov
571-272-7822

Paper 60
Entered: February 19, 2014

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

SYNOPSYS, INC.
Petitioner

v.

MENTOR GRAPHICS CORPORATION
Patent Owner

Case IPR2012-00042
Patent 6,240,376 B1

Before HOWARD B. BLANKENSHIP, SALLY C. MEDLEY, and JENNIFER S. BISK, *Administrative Patent Judges*.

BISK, *Administrative Patent Judge*.

FINAL WRITTEN DECISION

35 U.S.C. § 318(a) and 37 C.F.R. § 42.73

I. INTRODUCTION

A. Background

Petitioner, Synopsys, Inc. ("Synopsys"), filed a petition on September 26, 2012, for *inter partes* review of claims 1-15 and 20-33 of U.S. Patent No. 6,240,376 B1 ("the '376 Patent") pursuant to 35 U.S.C. §§ 311-319. Paper 1 ("Pet."). Patent Owner, Mentor Graphics Corporation ("Mentor Graphics"), filed a preliminary response on December 28, 2012. Paper 15 ("Prelim. Resp."). On February 22, 2013, the Board denied the petition as to claims 10, 12-15, 20-27, and 30-33, and instituted trial for claims 1-9, 11, 28, and 29, on one ground of unpatentability, anticipation by U.S. Patent No. 6,132,109 ("Gregory") (Ex. 1007). Paper 16 ("Decision to Institute").

After institution of trial, Mentor Graphics filed a patent owner response. Paper 28 ("PO Resp."). Mentor Graphics also filed a substitute motion to amend claims by submitting proposed new claims 34-43 for claims 1, 5, 28, 2, 3, 6, 8, 9, 11, and 29, respectively. Paper 31 ("Mot. to Amend"). Synopsys filed a reply to the patent owner response (Paper 36; "Reply"), and also an opposition to Mentor Graphics's motion to amend (Paper 35; "Opp."). Mentor Graphics then filed a reply in support of its motion to amend. Paper 39 ("Reply Mot. to Amend").

In preparation for oral hearing, both parties filed and fully briefed motions to exclude. Paper 42 ("Mentor Graphics's Motion to Exclude"); Paper 44 ("Synopsys's Motion to Exclude"). Oral hearing was held November 14, 2013. Paper 59 ("Transcript").

The Board has jurisdiction under 35 U.S.C. § 6(c). This final written decision is issued pursuant to 35 U.S.C. § 318(a) and 37 C.F.R. § 42.73.

Synopsys has shown that claims 5, 8, and 9 are unpatentable. Synopsys, however, has not met its burden to show by a preponderance of the evidence that claims 1-4, 6, 7, 11, 28, and 29 are unpatentable.

Mentor Graphics's motion to amend claims is *denied.*

B. The '376 Patent

The '376 patent generally relates to the fields of simulation and prototyping of integrated circuits. Ex. 1001, col. 1, ll. 10-11. In particular, the patent describes "debugging synthesizable code at the register transfer level during gate-level simulation." *Id.* at ll. 11-13.

As described in the Background of the Invention, integrated circuit design begins with a description of the behavior desired in a hardware description language ("HDL") such as Very High Speed Integrated Circuit Description Language ("VHDL"). *Id.* at ll. 14-25. A subset of HDL source code is referred to as Register Transfer Level ("RTL") source code. *Id.* at ll. 28-30. This RTL source code can be simulated using software, which typically offers robust debugging functionality for analyzing and verifying the design, including navigating the design hierarchy, viewing the RTL source code, setting breakpoints on a statement of RTL source code to stop the simulation, and viewing and tracing variables and signal values. *Id.* at ll. 44-54. However, although flexible, software RTL simulators are slow compared with hardware emulation. *Id.* at ll. 55-63. Thus, it often is desirable to use gate-level simulation to verify complex designs. *Id.*

The RTL description of a circuit can be used by synthesis tools to generate a "gate-level netlist," which, in turn, can be converted to a format suitable for programming a hardware emulator. *Id.* at ll. 35-42. A gate-level netlist represents the circuit to be simulated and ultimately is comprised of

combinatorial or sequential logic gates (e.g. AND, NAND, and NOR gates, or flip-flops and latches) and a description of their interconnections using signals (signals are also referred to as nets). *Id.* at col. 4, ll. 5-17. As discussed, gate-level simulation is useful for validation of a circuit design. *Id.* at col. 1, ll. 55-67. However, one disadvantage of gate-level simulation is that much of the high-level information from the RTL source code is lost during synthesis, resulting in debugging functionality that is limited severely in comparison with that available in software RTL simulation. *Id.* at col. 2, ll. 1-23.

The ’376 patent describes a method of synthesizing RTL source code such that the resulting gate-level simulation can support the traditional debugging tools of setting breakpoints, mapping signal values to particular source code lines, and stepping through the source code to trace variable values. *Id.* at ll. 1-30. The Summary of the Invention describes facilitating debugging during gate-level simulation by: (1) generating “instrumentation logic indicative of the execution status of at least one synthesizable statement within the RTL source code”; (2) generating a gate-level netlist from the RTL source code; and (3) during simulation, evaluating the instrumentation logic of the gate-level netlist to enable RTL debugging. *Id.* at ll. 26-39.

The ’376 patent describes two main embodiments for implementing this method. The first embodiment modifies the gate-level netlist to provide instrumentation signals “implementing the instrumentation logic and corresponding to synthesizable statements within the RTL source code.” *Id.* at ll. 40-43. This modification of the gate-level netlist can be done either by modifying the RTL source code directly or by generating the modified gate-

level netlist during synthesis. *Id.* at ll. 43-46. The second embodiment ("the cross-reference embodiment") describes storing the instrumentation signals in a cross-reference database instead of modifying the gate-level netlist. *Id.* at ll. 47-52.

Figure 2 of the '376 patent, reproduced below, illustrates "one embodiment of the instrumentation process in which instrumentation is integrated with the synthesis process." *Id.* at col. 5, ll. 9-11.

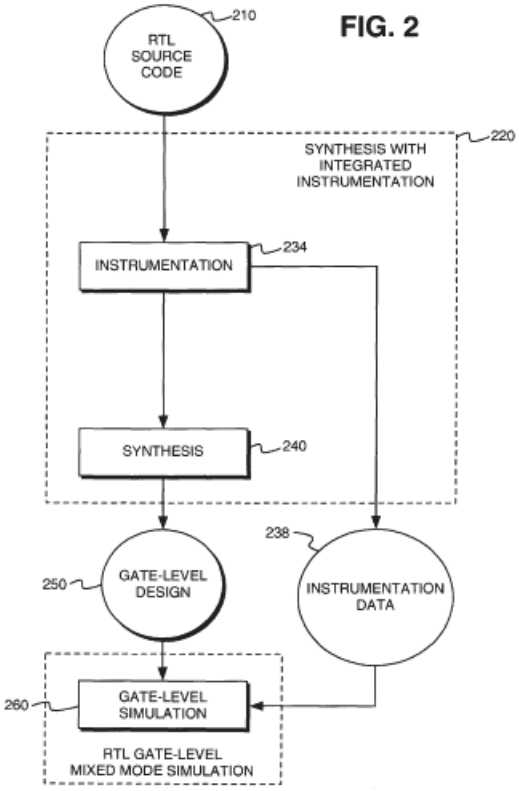


Figure 2, above, shows that RTL source code 210 is provided to synthesis process 220, which includes instrumentation step 234 followed by synthesis step 240. *Id.* at ll. 11-16. In the first embodiment, in which the gate level netlist is modified to include instrumentation signals, the resulting gate-level design 250 "contains additional logic to create the additional instrumentation output signals referenced in instrumentation data 238." *Id.* at ll. 17-30.

Instrumentation data 238 is implemented as gates that can then be simulated. *Id.* at col. 6, ll. 32-37.

In the cross-reference embodiment, "the RTL source code is analyzed to generate a cross-reference database as instrumentation data 238 without modifying the gate-level design." *Id.* at col. 5, ll. 31-33. In this embodiment, "[t]he instrumentation data 238 is likely to contain considerably more complex logic to evaluate during simulation." *Id.* at ll. 42-45.

The '376 patent describes tradeoffs between the two main embodiments. *Id.* at l. 45. For example, the first embodiment reduces the complexity of the logic to be evaluated during simulation, resulting in faster simulation time. *Id.* at ll. 46-64. However, because the gate-level design used during simulation is modified to accommodate the debugging logic, the design actually used for production will differ from that used during simulation, and, thus, the simulation may not reproduce accurately the production behavior of the circuit. *Id.* On the other hand, the cross-reference embodiment typically results in greater complexity of instrumentation logic to evaluate during simulation, resulting in longer simulation time. *Id.* at ll. 65-67. In addition, some of the evaluation may be performed by software, instead of hardware, eliminating direct verification of the target system through in-situ verification. *Id.* at col. 5, l. 65 – col. 6, l. 11. However, the technique does not affect the original gate-level design, and the instrumentation data can be eliminated after testing without disrupting the gate-level design. *Id.* Because of these various tradeoffs, the '376 patent mentions generally, but does not describe in detail, alternate

embodiments that combine the two main embodiments "in order to trade off simulation speed, density, and verification accuracy." *Id.* at col. 6, ll. 17-22.

The '376 patent subsequently describes (in Figures 3, 12, and 17 and the related text) three methods of modifying the gate-level netlist. *Id.* at col. 13, ll. 38-40. As described when discussing the first embodiment above, the '376 patent discloses that these three methods can be applied either by modifying the RTL source code directly by applying the method to the source code before it is synthesized independently of the synthesis process (*id.* at ll. 55-59), as shown in Figures 3, 12, and 17, or they can be integrated into the synthesis tool so that actual modification of the RTL source code is not required (*id.* at ll. 60-67).

Figure 3, reproduced below, illustrates a method of modifying RTL source code for sequential statements that depend only on the value of the inputs and can be synthesized to logic networks of combinatorial gates and latches ("level-sensitive RTL source code"). *Id.* at col. 7, ll. 13-22, 40-43.

FIG. 3

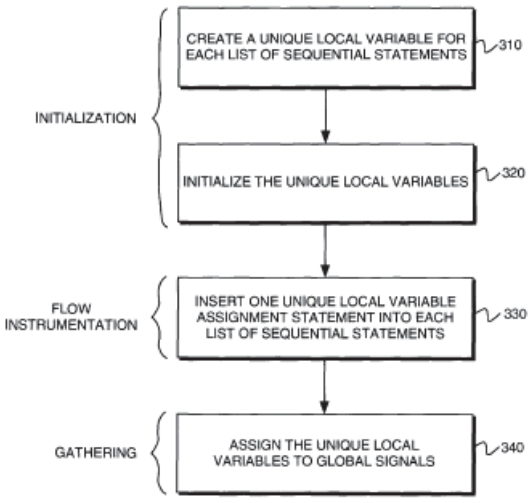
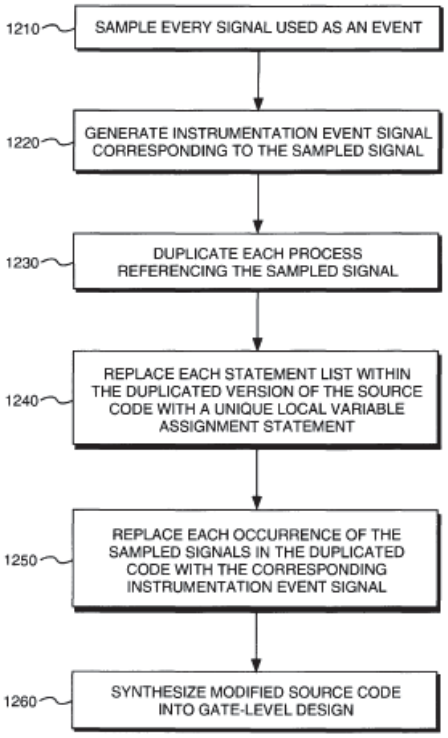


Figure 3, above, shows a method in which a unique local variable is created for each list of adjacent sequential statements in step 310, each of

these variables is initialized to zero in step 320, and one unique variable assignment statement is inserted into each list of adjacent sequential statements corresponding to an executable branch in step 330. *Id.* at ll. 40-50. At the end of the process, all the unique local variables are assigned to global signals in step 340. *Id.* at ll. 50-54.

Figure 12, reproduced below, illustrates a method of modifying RTL source code having references to signal events, typically used to describe edge-sensitive devices such as flip-flops. *Id.* at col. 9, ll. 27-32, 63-64.

FIG. 12



The method shown in Figure 12, above, begins with step 1210, in which every signal whose state transition serves as the basis for the determination of another signal is sampled. *Id.* at ll. 63-67. An instrumentation signal event is generated in step 1220, and every process that references a signal event is duplicated in step 1230. *Id.* at col. 9, l. 67 –

col. 10, l. 7. In step 1240 each list of sequential statements within the duplicate version of the code is replaced by a unique local variable assignment. In step 1250, each time a signal event is referenced in the duplicated version of the code, it is replaced by the sampled signal event computed in step 1210. *Id.* at col. 10, ll. 7-12. Finally, the RTL source code is synthesized, in step 1260, to generate gate-level logic, including the instrumentation signals. *Id.* at ll. 12-14.

Figure 17, reproduced below, illustrates a method of modifying RTL source code for processes themselves for subsequent determination of whether the process is active during gate-level simulation. *Id.* at col. 11, ll. 43-46.

FIG. 17

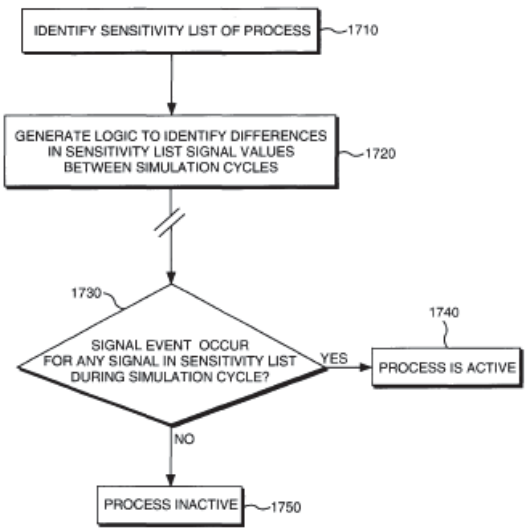


Figure 17, above, shows a method in which the sensitivity list of a process is identified in step 1710, logic is generated to compare the signals in the sensitivity list between consecutive simulation cycles in step 1720, and during gate-level simulation in step 1730, a determination is made as to whether an event has occurred on any of the sensitivity list signals. *Id.* at

ll. 48-53. If the signal indicates a difference during a simulation cycle, as indicated by step 1740, the process is active; otherwise, the process is inactive, as indicated by step 1750. *Id.* at ll. 53-58.

C. Illustrative Claims

Three of the claims involved in this proceeding, claims 1, 5, and 28, are independent. All three are reproduced below:

1. A method comprising the steps of:

a) identifying at least one statement within a register transfer level (RTL) synthesizable source code; and

b) synthesizing the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of the at least one statement.

5. A method of generating a gate level design, comprising the steps of:

a) creating an instrumentation signal associated with at least one synthesizable statement contained in a register transfer level (RTL) synthesizable source code; and

b) synthesizing the source code into a gate-level design having the instrumentation signal.

28. A storage medium having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code, wherein when executed the instructions enable the processor to synthesize the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of at least one synthesizable statement of the source code.

II. ANALYSIS

A. 35 U.S.C. § 315(b)

As a threshold issue, Mentor Graphics argues that this proceeding is barred by virtue of the relationship between Synopsys and the companies of Synopsys Emulation and Verification, S.A. and EVE-USA, Inc. (collectively "EVE"). PO Resp. 2-22. Mentor Graphics bases these arguments on the following facts.

Luc Burgun, a named inventor of the '376 patent, was, at one time, a Mentor Graphics employee. PO Resp. 3 (citing Ex. 2028: Ex. 5 at 1-4.[1] Burgun assigned all rights in the invention claimed in the '376 patent to Mentor Graphics. *Id.* (citing Ex. 2029: Ex. 2 at 3). Subsequently, Burgun left Mentor Graphics and went to work for EVE. *Id.* In 2006, Mentor Graphics filed suit against EVE in the United States District Court for the District of Oregon, alleging that EVE's ZeBu emulators infringed the '376 patent. *Id.* at 4 (citing Ex. 2001); *Mentor Graphics Corp. v. EVE-USA, Inc.*, 06-341-AA (D. Or. 2006). That case was dismissed with prejudice pursuant to a settlement agreement. Ex. 2003. Shortly after filing the petition in the present case, EVE and Synopsys jointly filed a declaratory judgment action in the United States District Court for the Northern District of California, seeking a ruling of non-infringement and invalidity of the '376 patent. Ex. 2004. The complaint states that "[o]n September 27, 2012, Synopsys, Inc. entered into an agreement to acquire the business of EVE," which acquisition "is expected to close in the immediate future." *Id.* at ¶ 13.

[1] Exhibits 2028 and 2029 are large exhibits, not paginated consecutively, including many non-sequentially numbered Exhibits. Throughout this Decision, citations to these exhibits will be of the form "Ex. 202[8 or 9]: [Ex. # within Ex. 202X at page number of that Ex. #]."

Mentor Graphics contends that the acquisition took place on October 4, 2012. PO Resp. 4-5 (citing Ex. 2029: Ex. 34 at 20).

1. Privity

Mentor Graphics argues that this *inter partes* review is barred because Synopsys and EVE were in privity at the time of the Decision to Institute. *Id.* at 6-7. Mentor Graphics asserts that based on this relationship the complaint served on EVE in the May 2006 case should trigger § 315(b). *Id.* at 6. We disagree with Mentor Graphics's contentions.

35 U.S.C. § 315(b) states as follows:

> An inter partes review may not be instituted if the petition requesting the proceeding is filed more than 1 year after the date on which the petitioner, real party in interest, or privy of the petitioner is served with a complaint alleging infringement of the patent. The time limitation set forth in the preceding sentence shall not apply to a request for joinder under subsection (c).

The Office promulgated a rule interpreting § 315(b), 37 C.F.R. § 42.101(b), which states that:

> A person who is not the owner of a patent may file with the Office a petition to institute an inter partes review of the patent unless:
> . . .
> (b) The petition requesting the proceeding is filed more than one year after the date on which the petitioner, the petitioner's real party-in-interest, or a privy of the petitioner is served with a complaint alleging infringement of the patent.

This rule makes clear that it is only privity relationships up until the time a petition is filed that matter; any later-acquired privies are irrelevant.

Furthermore, privity is a "flexible and equitable" doctrine rooted in common law. Office Patent Trial Practice Guide, 77 Fed. Reg. 48,756,

48,759 (Aug. 14, 2012). Consistent with this doctrine, we also take into consideration the nature of the relationship between the parties at the time that the statutorily-referenced complaint was served. *See, e.g.*, *Taylor v. Sturgell*, 553 U.S. 880, 892 (2008) ("A person who was not a party to a suit generally has not had a 'full and fair opportunity to litigate' the claims and issues settled in that suit."); *Mars Inc. v. Nippon Conlux Kabushiki-Kaisha*, 58 F.3d 616, 619 (Fed. Cir. 1995) (applying res judicata to parent corporation because it controlled wholly-owned subsidiary during prior litigation). Mentor Graphics has not alleged that Synopsys was a privy of EVE in 2006 when EVE was served with a complaint alleging infringement of the '376 patent. Thus, there is no contention that Synopsys had any control of this previous suit or even had notice of it, along with an opportunity to participate while it was still pending. *See Richards v. Jefferson Cnty., Ala.*, 517 U.S. 793 (1996) (holding no estoppel where subsequent plaintiffs were not provided notice of first suit nor adequately represented in it). Thus, this lack of relationship between Synopsys and EVE in the 2006 litigation is another reason to conclude that there was no privity relationship between Synopsys and EVE sufficient to trigger § 315(b)'s prohibitions.

Moreover, no record evidence suggests that Synopsys's petition for review was timed to inject delay into an already-pending litigation and, thus, this case does not implicate the concerns that this statute appears designed to address. H.R. REP. NO. 112-98, at 45 (2011) (explaining § 315(b) as "*Time limits during litigation.* Parties who want to use inter partes review during litigation are required to seek a proceeding within 12 months of being served with a complaint alleging infringement of the patent."); 157 CONG. REC.

S1326 (daily ed. Mar. 7, 2011) (statement of Sen. Sessions) ("The bill also includes many protections that were long sought by inventors and patent owners . . . . It imposes time limits on starting an inter partes or post-grant review *when litigation is pending* . . . . All of these reforms will help to ensure that post-grant review operates fairly and is not used for purposes of harassment or delay." (emphasis added)).

Thus, we conclude that there was no privity relationship between Synopsys and EVE sufficient to trigger § 315(b)'s prohibitions, and we decline to dismiss the *inter partes* review on this basis.

2. Real Party-in-interest

Mentor Graphics argues that this *inter partes* review is barred because EVE is a real party-in-interest to this *inter partes* review. PO Resp. 7-14. Mentor Graphics asserts that, therefore, the complaint served on EVE in the May 2006 case should trigger § 315(b). *Id*. at 6. Mentor Graphics admits that "on the date the petition for this [*inter partes* review] was filed, Synopsys had not yet acquired EVE and therefore had at best merely a prospective interest in the ZeBu products." *Id.* at 9. Mentor Graphics, however, asserts that because Synopsys had a prospective interest in invalidating the '376 patent when the petition was filed, "Synopsys was acting as an agent for the benefit of EVE." *Id.* at 10. Thus, according to Mentor Graphics, at the time of filing, Synopsys was a third-party beneficiary for whose benefit the action was brought and, therefore, a real party-in-interest. *Id.* at 7-10.

Mentor Graphics also contends that Synopsys allowed EVE to direct or control content of the petition for this *inter partes* review because Synopsys (1) specifically acquired EVE because of its expertise in the

technology field to which the '376 patent is directed; (2) jointly asserted with EVE the same non-infringement and invalidity claims and defenses with respect to the '376 patent, using the same counsel, in the jointly-filed declaratory judgment litigation; and (3) planned and coordinated the timing of the filing of the petition in this case and the declaratory judgment complaint with EVE. PO Resp. 12-13. Based on these contentions, Mentor Graphics asserts that Synopsys and EVE "conspired together to conceal EVE's status as a 'real party-in-interest' to circumvent and thwart the statutory estoppel provisions." *Id.* at 14.

As discussed above, 37 C.F.R. § 42.101(b) makes clear that it is only relationships up until the time a petition is filed that matter. Mentor Graphics does not point to persuasive evidence to support its assertions that Synopsys allowed EVE to direct or control content of the petition filed in this case or any other evidence that EVE was a real party-in-interest prior to the filing of the petition. Although Mentor Graphics filed a Motion for Additional Discovery on the topic of real party-in-interest (Paper 21), this motion was denied because it did not articulate clearly why such discovery was "necessary in the interest of justice" as required by 35 U.S.C. § 316(a)(5) (Paper 24). In fact, the entirety of Mentor Graphics's explanation of why it needed additional discovery on the subject of real party-in-interest was the following:

> Thus, while the request interest of justice, [sic] as required by 37 C.F.R. § 42.51(b)(2), in order to allow the Patent Owner an opportunity to show the applicability of a § 315(b) bar under the legal standard adopted by the Board. This includes the opportunity to show further (1) . . . (4) the status of EVE as a real party-in-interest to this IPR.

Paper 21 at 2-3.

Thus, because Mentor Graphics has not supported sufficiently its assertions that EVE was a real party-in-interest at the time the petition in this case was filed, we decline to dismiss the *inter partes* review on this basis.

B. Assignor Estoppel

Mentor Graphics argues that Synopsys is barred from challenging the validity of the '376 patent by assignor estoppel. PO Resp. 14-22. The Board has determined previously, and we agree, that assignor estoppel is not a basis for denying a petition requesting *inter partes* review:

> Under the AIA, "a person *who is not the owner of a patent* may file with the Office a petition to institute an inter partes review of the patent." 35 U.S.C. § 311(a) (emphasis added). Consequently, under the statute, an assignor of a patent, who is no longer an owner of the patent at the time of filing, may file a petition requesting *inter partes* review. This statute presents a clear expression of Congress's broad grant of the ability to challenge the patentability of patents through *inter partes* review.

Athena Automation Ltd. v. Husky Injection Molding Sys. Ltd., IPR2013-00290, slip op. at 12-13 (PTAB Oct. 25, 2013), Paper No. 18; *see also Palo Alto Networks, Inc. v. Juniper Networks, Inc.*, IPR2013-00369, slip op. at 11-14 (PTAB Dec. 19, 2013), Paper No. 16.

Mentor Graphics further asserts that even if Synopsys is not barred from requesting *inter partes* review, the Board should exercise its discretion to dismiss this *inter partes* review because of the relationship between Mr. Burgun and Synopsys.[2] PO Resp. 16-19. Mentor Graphics further argues,

[2] This case does not require us to reach the issue of whether Mr. Burgun is in privity with Synopsys, as asserted by Mentor Graphics (PO Resp. 16-19), because we conclude that even if Mentor Graphics established such a relationship, Mentor Graphics has not shown a sufficient basis to bar

more generally, that equitable considerations weigh against granting the petition, including that Synopsys is in privity with EVE and shares personnel with EVE, including Mr. Burgun. *Id.* at 19. Moreover, according to Mentor Graphics, "[i]t would be wholly against the principles of assignor estoppel to allow Synopsys to receive the benefit of the acquisition of EVE, but avoid EVE's equitable obligations." *Id.* at 22.

We are not persuaded, however, that the equitable doctrine of assignor estoppel provides an exception to the statutory mandate that any person who is not the owner of a patent may file a petition for *inter partes* review. Accordingly, we decline to dismiss the *inter partes* review based on Mentor Graphics's estoppel arguments.

C. Claim Construction

In an *inter partes* review, claim terms in an unexpired patent are interpreted according to their broadest reasonable construction in light of the specification of the patent in which they appear. 37 C.F.R. § 100(b); Office Patent Trial Practice Guide, 77 Fed. Reg. 48,756, 48,766 (Aug. 14, 2012). Claim terms are also given their ordinary and customary meaning, as would be understood by one of ordinary skill in the art in the context of the entire disclosure. *In re Translogic Tech., Inc.*, 504 F.3d 1249, 1257 (Fed. Cir. 2007).

If an inventor acts as his or her own lexicographer, the definition must be set forth in the specification with reasonable clarity, deliberateness, and precision. *Renishaw PLC v. Marposs Societa' per Azioni*, 158 F.3d 1243, 1249 (Fed. Cir. 1998). The construction that stays true to the claim language

---

Synopsys from further participation in, or dismissal of, this *inter partes* review.

and most naturally aligns with the inventor's description is likely the correct interpretation. *Id.* at 1250.

1. "Instrumentation Signal"

Construction of the term "instrumentation signal," required by each of the claims at issue in this proceeding, is central to the patentability determination. *See* PO Resp. 29-37; Reply 2-8. For example, claims 1 and 28 recite "including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status." Claim 5, the third, and final, challenged independent claim, recites "creating an instrumentation signal associated with at least one synthesizable statement contained in a register transfer level (RTL) synthesizable source code."

a. Construction Adopted in the Decision to Institute

Although neither the petition nor the preliminary response set forth a specific construction for "instrumentation signal," the Board adopted, for purposes of the Decision to Institute, an interpretation that "the claimed instrumentation signal at least encompasses an output signal created during synthesis of RTL source code by inserting additional logic, preserved from the source code, that indicates whether the corresponding RTL source code statement is active." Decision to Institute 10.

Mentor Graphics argues that this construction is only partially correct. PO Resp. 30. Specifically, Mentor Graphics agrees that "instrumentation signals" encompass "inserting additional logic." *Id.* According to Mentor Graphics, however, the requirement that the "additional logic is preserved from the source code" is contrary to how one of ordinary skill in the art would understand the term in light of the specification. *Id.* at 35 (citing Ex. 2027 ¶¶ 38-39). Mentor Graphics points to language in the '376 patent

specification stating that "[i]nstrumentation is the process of preserving high-level information through the synthesis process." PO Resp. 35-36 (quoting Ex. 1001, col. 5, ll. 3-4). Mentor Graphics's expert, Dr. Majid Sarrafzadeh, states that preserving information here refers to permitting relation back from the execution of the gate level netlist to the corresponding statements in the RTL source code, not the preservation of *logic* itself from the source code. Ex. 2027 ¶ 40.

We find this argument, along with the supporting evidence, persuasive. The language of the '376 patent also supports this conclusion. For example, immediately following the language quoted above, the '376 specification states that "[i]nstrumentation permits simulation of a gatelevel netlist at the level of abstraction of RTL simulation by *preserving some of the information* available at the source code level through the synthesis process." Ex. 1001, col. 5, ll. 4-8 (emphasis added).

b. Definition of "Instrumentation"

In proposing an alternative construction for "instrumentation signal," Mentor Graphics initially asserts that the customary meaning of the term "instrumentation" to those of skill in the art is "additional code inserted into a program to monitor and/or collect information about the program behavior or operation during program execution." PO Resp. 31. Dr. Sarrafzadeh testifies that this is how the term is "generally recognized and understood in the programming language arts." Ex. 2027 ¶ 30 (citing IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990 at 41 ("Devices or instructions installed or inserted into hardware or software to monitor the operation of a system or component."); National Bureau of Standards [NBS] Special Publication 500-75 Validation, Verification, and

Testing of Computer Software at 48 (1981) ("The insertion of additional code into the program in order to collect information about program behavior during program execution.")). We agree that this is the ordinary and customary meaning of the first word—"instrumentation"—of the term "instrumentation signal."

c. Specification

Mentor Graphics further asserts that the term "instrumentation signal" requires that "the signal be provided by logic that is *additional* to the design logic resulting from the synthesis of the RTL source code." PO Resp. 30. In other words, the instrumentation signal cannot be created solely by preserving circuit components.

Synopsys argues that Mentor Graphics's construction is too narrow and that the broadest reasonable construction of "instrumentation signal" is broad enough to include creation solely using preservation of circuit components. Reply 4-8.

According to Mentor Graphics, one of ordinary skill would understand the following excerpts of the specification "to effectively define" "instrumentation signal" to require that instrumentation logic be added to the gate-level netlist (PO Resp. 31-32):

> Generally *instrumentation logic* is created for a synthesizable statement in the RTL source code either by modifying the RTL source code or by analyzing the RTL source code during the synthesis process. The *instrumentation logic* provides an output signal indicative of whether the corresponding synthesizable statement is active. A gate-level design including the instrumentation output signal is then synthesized. Referring to FIG. 2, *the resulting gate-level design 250 contains additional logic to create the additional instrumentation output signals* referenced in instrumentation data 238.

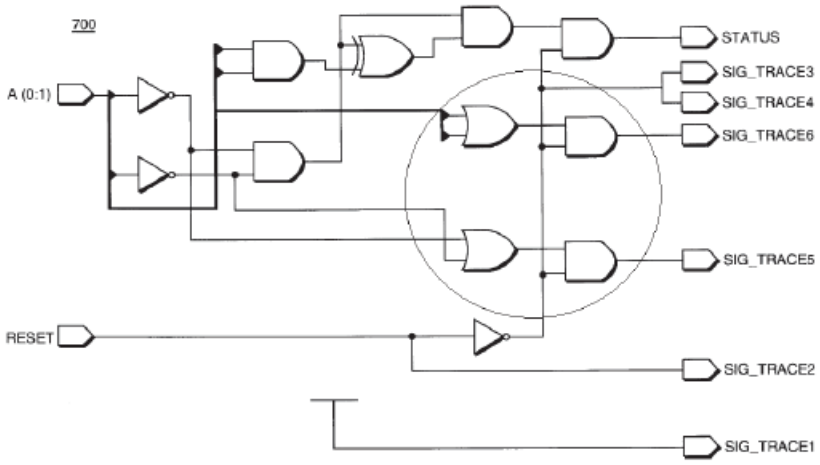
Ex. 1001, col. 5, ll. 21-30 (emphasis added by Mentor Graphics).

> FIG. 7 illustrates one embodiment of the logic 700 generated through instrumentation. In particular, FIG. 7 illustrates the *additional gate-level logic added* to generate signals SIG_TRACE1 through SIG_TRACE6 from synthesis of the modified source code.

Id. at col. 8, ll. 60-64 (emphasis added by Mentor Graphics). We are not persuaded that the quoted language qualifies as a limiting definition of "instrumentation signal." "To be his own lexicographer, a patentee must use 'a special definition of the term [that] is clearly stated in the patent specification or file history.'" *Laryngeal Mask Co. v. Ambu*, 618 F.3d 1367, 1372 (Fed. Cir. 2010) (quoting *Vitronics Corp. v. Conceptronic, Inc.*, 90 F.3d 1576, 1580 (Fed. Cir. 1996)). The specification here does not indicate clearly the patentee's intent to define or limit the definition of "instrumentation signal." Indeed, the first excerpt relied upon by Mentor Graphics is extracted from a paragraph that begins: "In one embodiment . . . ." Similarly, the second excerpt begins by stating that it "illustrates one embodiment." Thus, all of the language relied upon by Mentor Graphics as "effectively defin[ing]" the term, refers only to illustrative examples. It is well-settled that when claim language is broader than the preferred embodiment, the claims are not to be confined to that embodiment. *DSW, Inc. v. Shoe Pavilion, Inc.*, 537 F.3d 1342, 1348 (Fed. Cir. 2008). Moreover, "just as the preferred embodiment itself does not limit claim terms, mere inferences drawn from the description of an embodiment of the invention cannot serve to limit claim terms, as they are insufficient to require a narrower definition of a disputed term." *Johnson Worldwide Assocs., Inc. v. Zebco Corp.*, 175 F.3d 985, 992 (Fed. Cir. 1999) (internal citations omitted).

Mentor Graphics adds that Figure 7, which "illustrates the additional gate-level logic added to generate signals SIG_TRACE1 through SIG_TRACE6 from synthesis of the modified source code [shown in Figure 6]" (Ex. 1001, col. 8, ll. 61-64), supports its narrow construction. Dr. Sarrafzadeh testifies that this can be shown with an annotated copy of Figure 7, reproduced below.

FIG. 7



Ex. 2027 ¶ 33. The annotated copy of Figure 7, above, illustrates gate-level logic synthesized from modified RTL source code. Ex. 1001, col. 3, ll. 29-30. Dr. Sarrafzadeh testifies that the circle was added to emphasize the instrumentation logic (two AND gates and two OR gates) added to the original circuit shown in Figure 5. Ex. 2027 ¶ 33.

As pointed out by Synopsys (Reply 6), however, this Figure actually supports a broader construction of "instrumentation signal." The additional logic within the marked circle is required only for four (SIG_TRACE3, SIG_TRACE4, SIG_TRACE5, and SIG_TRACE6) of the six instrumentation signals included in the figure. SIG_TRACE1 and SIG_TRACE2 are shown with no additional logic gates. Mentor Graphics

attempts to explain away this incongruity in its argument by stating that the specification itself is incorrect. For example, at the final hearing, in response to the question "[w]hat you're telling us about additional logic, is it consistent with the patent at column 9 when it talks about Figure 10 where it calls SIG_TRACE1 and SIG_TRACE2 instrumentation signals?" counsel for Mentor Graphics stated:

> The patent with respect to Figures 7 and 10 calls all of the added signals instrumentation signals, that is correct, but we do not believe that all of those are instrumentation signals as that term should be included. It was an unfortunate, but expedient way to group all of the signals that have been added to Figure 7 and Figure 10. Some of those signals do not, however, provide any additional information. First, they're not logic.

Paper 59 (Transcript) 28-29. We are not persuaded that the term "instrumentation signal" should be limited such that it is inconsistent with explicit statements in the specification.

d. Alternate Embodiment

In addition, Mentor Graphics argues that the challenged claims should be construed to exclude the cross-reference embodiment. PO Resp. 33. The '376 patent explicitly states that in the cross-reference embodiment, "the gate-level netlist is not modified but the instrumentation signals implementing the instrumentation logic are contained in a cross-reference database." Ex. 1001, col. 2, ll. 47-50. Moreover, in both of the main embodiments, "instrumentation signals indicate the execution status of the corresponding cross-referenced synthesizable statement." *Id.* at ll. 50-52. Thus, in order to limit the definition of "instrumentation signal" as used in claims 1, 5, and 28 to signals created by additional instrumentation logic added to the gate-level netlist, Mentor Graphics contends that the challenged

claims exclude this embodiment. PO Resp. 33. Specifically, Dr. Sarrafzadeh testifies that "[c]laims 1, 5 and 28 do not describe [the cross-reference] embodiment because in this embodiment, rather than adding instrumentation signals (and associated logic) to the specified circuit, an external 'cross-reference database' uses 'already existing signals' to evaluate whether a particular line of source code is active." Ex. 2027 ¶ 35 (citing Ex. 1001, col. 5, ll. 31-37). This argument is circular, however, and does not explain why claims 1, 5, and 28 require additional logic be added to the gate-level netlist.

Moreover, during cross-examination, Dr. Sarrafzadeh admitted to not understanding the details of how the alternate cross-reference embodiment would work. For example, when questioned about language in the specification stating that "[w]ith respect to source code analysis, cross-reference instrumentation data including the instrumentation signals can be used to count the number of times a corresponding statement is executed in the source code" (Ex. 1001, col. 2, l. 66 – col. 3, l. 2), Dr. Sarrafzadeh answered that the '376 patent does not "show a method for doing this, [the patent] just draw[s] certain conclusions, and without showing and describing these designs, I would not have—it would not be clear what they mean here." Ex. 1019, 28:19 – 29:13. Further, Dr. Sarrafzadeh stated that his conclusions related to the cross-reference embodiment were based on his determination that the specification did not explain in detail how it would be implemented. *Id.* at 21:23 – 22:25.

It is true that "claims need not be construed to encompass all disclosed embodiments when the claim language is clearly limited to one or more embodiments." *TIP Sys., LLC v. Phillips & Brooks/Gladwin, Inc.*, 529 F.3d

1364, 1375 (Fed. Cir. 2008). Here, the claim language is broad enough to include the alternative cross-reference embodiment. Moreover, the written description does not evidence a clear intent to limit the invention to a singular embodiment. In fact, as described above, the specification describes two main embodiments, and multiple alternate embodiments that combine the two main embodiments. In fact, the first embodiment of the '376 patent is described in the most detail; however, "although the specifications may well indicate that certain embodiments are preferred, particular embodiments appearing in a specification will not be read into the claims when the claim language is broader than such embodiments." *Electro Med. Sys., S.A. v. Cooper Life Scis., Inc.*, 34 F.3d 1048, 1054 (Fed. Cir. 1994). In the present case, neither the claim language nor the specification suggests limiting the scope of the subject matter to a single embodiment.

e. Examiner's Reasons for Allowance

Mentor Graphics also points to the following language in the "'examiner's statement of reasons for allowance' in application (SN 09/127,584) that issued as the '376 patent" (PO Resp. 33):

> The current invention allows for the *insertion of instrumentation logic* which executes to function as a simulation breakpoint to be applied to gate-level circuit simulation. This is done by *inserting instrumentation points into the Register Transfer Level (RTL) design*, which can then be synthesized to the gate-level description . . . . The current invention extends [the higher level debugging of cited prior art] to a lower level in the design process, namely in the gate-level description that has been synthesized from the RTL description. This advantageously allows for the use of simulation breakpoints implemented by *inserting "instrumentation logic*

> *into the RTL description* (as indicated in the specification pg. 42, line 24) in a gate-level circuit simulation.

Ex. 2027 ¶ 35 (quoting Ex. D) (underlining in original; italics added by Mentor Graphics). We are not persuaded by this argument. The language quoted from the reasons for allowance makes it clear that the invention *allows for*, but does not state that the invention *requires*, additional logic to be added to the gate-level netlist resulting from the synthesis of the RTL source code. Moreover, Mentor Graphics neglects to include, in the quoted language, the last sentence of the paragraph, which states: "This provides a distinct advance over the prior art methods of matching input vectors to output vectors and attempting to back out the fault based only on these vectors." Ex. 2027: Ex. D at 106. This last sentence clarifies that the Examiner's statements were directed to distinguishing the claimed process of using instrumentation signals from the prior art methods of comparing input and output vectors, not to providing a definition of "instrumentation signal" or indicating that the instrumentation signal cannot be created solely using preservation of circuit components.

f. Definition of "Instrumentation Signal"

Thus, although we agree with Mentor Graphics that the definition of "instrumentation signal" does not require that additional logic inserted during synthesis of RTL source code is "preserved from the source code," we do not agree that "instrumentation signal" excludes creation by preserving already existing circuit components. Instead, we determine that the broadest reasonable construction of "instrumentation signal," in light of the '376 patent specification, at least includes an output signal created during synthesis of RTL source code by inserting additional code into a

program that indicates whether the corresponding RTL source code statement is active.

2. "Execution Status"

The term "execution status" is recited in claims 1-4, 28, and 29. Mentor Graphics states that "in the HDL context, 'execution status' refers to information regarding whether particular HDL statements have been executed or not." PO Resp. 37 (citing Ex. 2027 ¶ 41). This definition is consistent with the use of the term in the '376 patent. For example:

> In some cases the execution status of each branch of the code can be determined even though every branch is not explicitly instrumented. To verify the execution status of every branch, the instrumentation process need only ensure that each branch is instrumented either explicitly or implicitly through the instrumentation of other branches.

Ex. 1001, col. 12, ll. 33-38.

Synopsys does not address explicitly this proposed definition.

We determine that the broadest reasonable construction of "execution status" in light of the '376 patent specification is information regarding whether a particular HDL instruction has been performed.

D. Alleged Anticipation by Gregory under 35 U.S.C. § 102(e)

Synopsys asserts that claims 1-9, 11, 28, and 29, are unpatentable under 35 U.S.C. § 102(e) as anticipated by Gregory. As discussed above, claims 1, 5, and 28 are independent. Claims 2, 3, and 4 depend, directly or indirectly, from claim 1. Claims 6-9 and 11 depend, directly or indirectly, from claim 5. Claim 29 depends from claim 28.

To establish anticipation, each and every element in a claim, arranged as is recited in the claim, must be found in a single prior art reference. *Net MoneyIN, Inc. v. VeriSign, Inc.*, 545 F.3d 1359, 1369 (Fed. Cir. 2008);

Karsten Mfg. Corp. v. Cleveland Golf Co., 242 F.3d 1376, 1383 (Fed. Cir. 2001).

We have reviewed Synopsys's anticipation argument and supporting evidence, including Gregory's disclosure and the detailed claim chart appearing on pages 31-43 of the Petition. The claim chart persuasively reads all elements of each of claims 5, 8, and 9 onto the disclosure of Gregory. Despite the counter-arguments in Mentor Graphics's Patent Owner Response, and the evidence cited therein, which we also have considered, Synopsys has shown, by a preponderance of the evidence, that each of claims 5, 8, and 9 is unpatentable under 35 U.S.C. § 102(e) as anticipated by Gregory. *See* 35 U.S.C. § 316(e).

For claims 1-4, 6, 7, 11, 28, and 29, however, we give significant weight to the testimony of Mentor Graphics's expert, Dr. Sarrafzadeh, who persuasively explains that Gregory does not disclose each and every element of the claims. Synopsys does not provide sufficient evidence to rebut this testimony. Thus, as described below, we conclude that Synopsys has not met its burden to show by a preponderance of the evidence, as required by 35 U.S.C. § 316(e), that claims 1-4, 6, 7, 11, 28, and 29 are unpatentable.

1. Gregory (Ex. 1007)

Gregory "relates to . . . debugging digital circuits constructed using logic or behavioral synthesis." Ex. 1007, col. 1, ll. 12-15. Gregory describes a method of "providing a designer with the ability to mark the synthesis source code in the places that the designer wants to be able to debug" by "mark[ing] the source code with a particular text phrase, such as 'probe[,'] along with some additional optional information." *Id.* at col. 8, ll. 21-26. Further, Gregory describes a translation process that "interjects

information into the netlist" when it encounters a probe statement and "generates a circuit th[at] provides the same function as it did without the 'probe' statement, but adds additional information or components to the initial circuit that indicate that certain components should not be replaced during optimization" (col. 8, ll. 26-30). These components "are traceably related to the source HDL" and they "facilitate[] debugging." *Id.* at col. 8, ll. 35-41.

2. Claims 1-4, 28, and 29

Synopsys asserts that Gregory discloses claim 1's limitation of "synthesizing the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of the at least one statement." Pet. 31-32. Synopsys explains that "[s]ynthesizing the FIG. 8 VHDL code produces the gate-level circuit illustrated in FIG. 9, which includes the instrumentation signal tempout, which indicates an execution status of the instrumented statement." *Id.* at 32. Figures 8 and 9 of Gregory are reproduced below.

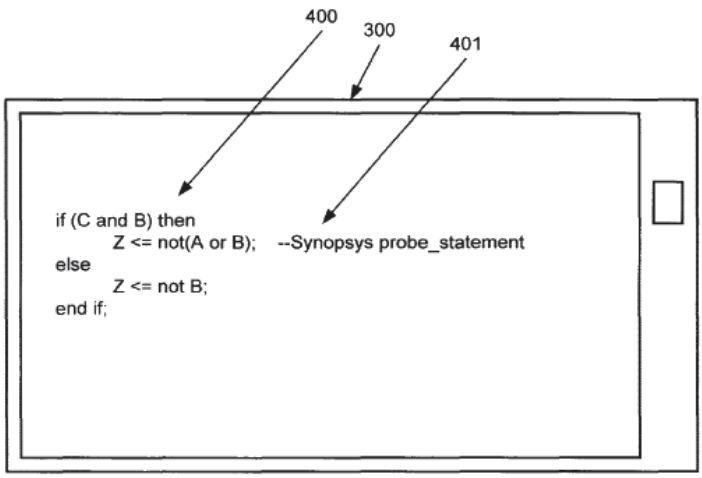


Figure 8, above, illustrates VHDL source code with a probe inserted.

Ex. 1007, col. 9, ll. 55-56. Gregory explains that the probe is "a directive to the translator" that "indicates that this particular VHDL statement should be processed so that it will be possible to relate analysis information to this point in the circuit." *Id*. at col. 12, ll. 54-61.

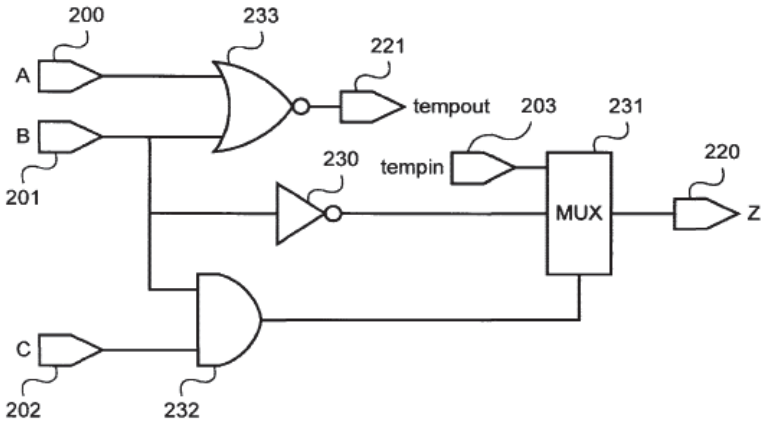


Figure 9, above, illustrates a circuit that a translator could produce using the source code with probe inserted shown in Figure 8. *Id*. at ll. 62-67. As discussed, Synopsys points to "tempout" as constituting the instrumentation signal. Pet. 32.

Mentor Graphics argues that tempout does not equate to the claimed "instrumentation signal" and Synopsys has not shown that Gregory discloses an "instrumentation signal, wherein the instrumentation signal is indicative of an execution status of the at least one statement."

Mentor Graphics argues that Gregory fails to disclose the claimed "instrumentation signal" because "tempout" does not involve additional logic, but instead simply preserves identified circuit nets during the optimization process. PO Resp. 41-46. This argument, however, requires a narrower construction of the term "instrumentation signal" than we have adopted, as discussed above. Thus, we agree with Synopsys that "tempout" qualifies as an "instrumentation signal" as we have construed that term.

Mentor Graphics also argues that "tempout" is not "indicative of an execution status," as required by claim 1. *Id.* at 46-50. Mentor Graphics points out that, in its petition, the only element Synopsys describes with particularity as disclosing this particular limitation of claim 1 is the "tempout" signal of Figure 9. *Id.* at 49-50. Dr. Sarrafzadeh testifies that in Figure 9, the result of the "tempout" signal is not indicative of the execution status of the HDL statement identified in Figure 8. Ex. 2027 ¶ 73. According to Dr. Sarrafzadeh, "tempout" reflects the result of "not (A or B)," while the HDL statement identified in Figure 8 is "Z<=not (A or B)," executed conditionally based on whether the "if" condition (if (C and B) then)) is true. *Id.* at ¶¶ 73-74. The execution status of the HDL statement thus depends on whether the "if" condition is true or not, but "tempout" does not indicate this information. *Id.*

In response to this argument, Synopsys asserts that Mentor Graphics improperly is looking solely at the 'tempout' signal and, in doing so, excluding an alternate embodiment also described in Gregory. Reply 10-11. Specifically, Synopsys argues that because execution status of every branch can be verified implicitly and not every branch need be instrumented, Mentor Graphics's argument is incorrect. *Id.* at 11 (citing Ex. 1001, col. 12, ll. 33-38). According to Synopsys, it is not necessary to determine the "C and B" condition in the "if" statement of Figure 8 to meet the "execution status" requirement. *Id.* Synopsys, however, does not point to any expert testimony to support these statements.

We give substantial weight to Mentor Graphics's expert testimony. Dr. Sarrafzadeh explains, persuasively, that the "tempout" signal is not "indicative of an execution status of the at least one statement" as required

by claim 1, where "execution status" is information regarding whether the particular source code instruction, identified in Figure 8, has been performed. *See* Ex. 2027 ¶¶ 73-78. Synopsys's unsupported argument to the contrary is not persuasive. Moreover, Synopsys only points to "tempout" in its petition (Pet. 32; 42-43) as disclosing this element of claim 1. Whether or not a particular embodiment requires the execution status of every branch to be verified explicitly, the claim language of claim 1 requires an "instrumentation signal, wherein the instrumentation signal is indicative of an execution status of the at least one statement." Synopsys has not shown persuasively that the "tempout" signal of Figure 9 discloses this limitation.

Gregory does not state explicitly that "tempout," or any other element, indicates an "execution status" of an HDL instruction. Although, as discussed, Gregory states that a probe "is a directive to the translator" that "indicates that this particular VHDL statement should be processed so that it will be possible to relate analysis information to this point in the circuit," (col. 12, ll. 54-61), we agree with Mentor Graphics that Synopsys does not point to any disclosure in Gregory that "analysis information" includes "execution status." PO Resp. 47. In fact, when Gregory discusses "analysis," it generally refers to characteristics of the circuit, such as timing and power, which are not related to "execution status" as we have construed that term. *See, e.g.*, Ex. 1007, col. 7, l. 59 – col. 8, l. 10; col. 12, ll. 49-61; col. 16, ll. 25-50.

Alternatively, Synopsys argues that Gregory *does* indicate the execution status of the "if" condition at the output of AND gate 232 in Figure 9. Reply at 11. Here, Synopsys appears to be arguing that B and C,

as opposed to the "tempout" signal, are the claimed "instrumentation signals." *Id.* ("Any argument that signals B and C cannot be instrumentation signals ignores the alternate embodiment which specifically contemplates implicit instrumentation and the use of already existing signals to determine execution status.").

We are not persuaded by this argument. To the extent that Synopsys is arguing that an element other than "tempout" discloses this limitation of claim 1, the argument is presented for the first time in Synopsys's reply and is not responsive to arguments made in Mentor Graphics's response. 37 CFR 42. 23(b). Moreover, Synopsys does not point to any evidence or persuasive argument to explain how B and C disclose the claimed instrumentation signal.

Synopsys has not shown that Gregory discloses the limitation "instrumentation signal, wherein the instrumentation signal is indicative of an execution status of the at least one statement." Thus, we conclude that Synopsys has not met its burden to show, by a preponderance of the evidence, that Gregory anticipates independent claim 1. *See* 35 U.S.C. § 316(e). Because independent claim 28 includes the same limitation, Synopsys also has not met its burden to show, by a preponderance of the evidence, that Gregory anticipates independent claim 28. For the same reasons, we conclude that Synopsys has not met its burden with regard to claims 2-4 and 29, which depend, either directly or indirectly, from claims 1 and 28.

3. Independent Claim 5

Independent claim 5 includes the limitation "creating an instrumentation signal associated with at least one synthesizable statement

contained in a register transfer level (RTL) synthesizable source code." Synopsys points to the analysis of claim 1 to show that this limitation is anticipated by Gregory. Pet. 34. In addition, to support its argument, Synopsys points to Gregory's Figures 12, 16, and 18 and the associated text describing a "block probe methodology for instrumenting all of the signals within a process statement." *Id.* at 34-35. Specifically, Synopsys asserts that the "signals 'temp_out' in FIG. 18 are instrumentation signals associated with the VHDL (RTL) statements within the instrumented 'process.'" *Id.* at 35. Figures 12, 16, and 18 are reproduced below.

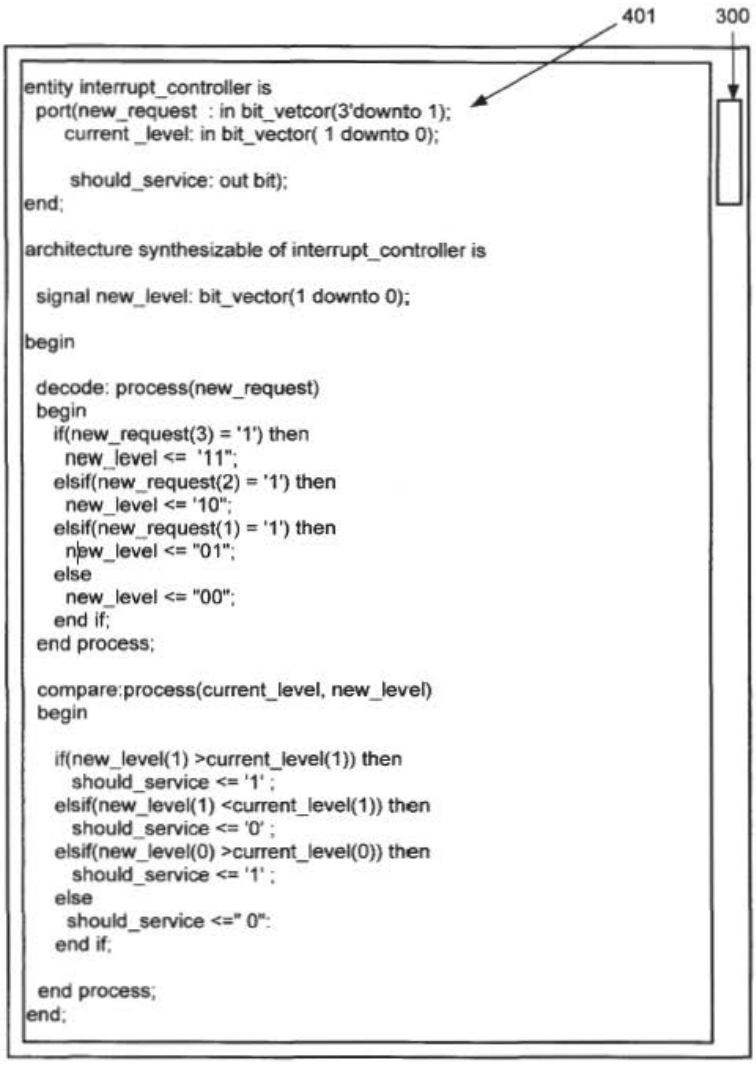
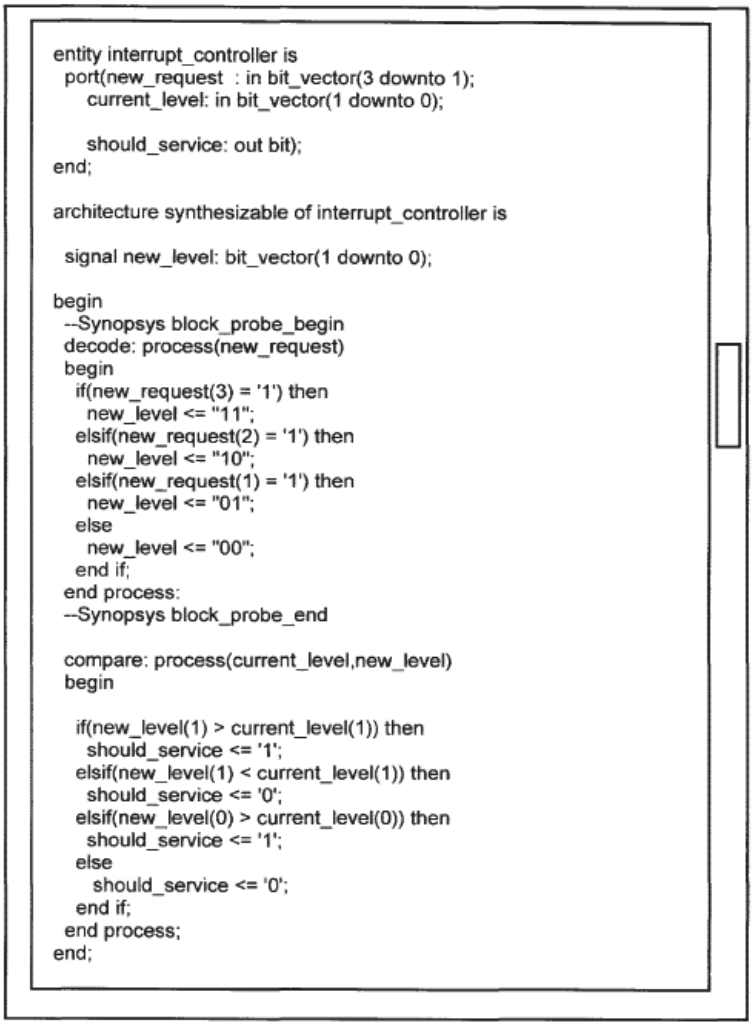


Figure 12

Figure 12, above, illustrates VHDL source code without probes using two process blocks. Ex. 1007, col. 9, ll. 64-65.



```
entity interrupt_controller is
  port(new_request  : in bit_vector(3 downto 1);
     current_level: in bit_vector(1 downto 0);

     should_service: out bit);
end;

architecture synthesizable of interrupt_controller is

  signal new_level: bit_vector(1 downto 0);

begin
  --Synopsys block_probe_begin
  decode: process(new_request)
  begin
    if(new_request(3) = '1') then
      new_level <= "11";
    elsif(new_request(2) = '1') then
      new_level <= "10";
    elsif(new_request(1) = '1') then
      new_level <= "01";
    else
      new_level <= "00";
    end if;
  end process;
  --Synopsys block_probe_end

  compare: process(current_level,new_level)
  begin

    if(new_level(1) > current_level(1)) then
      should_service <= '1';
    elsif(new_level(1) < current_level(1)) then
      should_service <= '0';
    elsif(new_level(0) > current_level(0)) then
      should_service <= '1';
    else
       should_service <= '0';
    end if;
  end process;
end;
```

Figure 16

Figure 16, above, illustrates the VHDL source code of Figure 12 with two block probes installed. *Id.* at col. 10, l. 7.

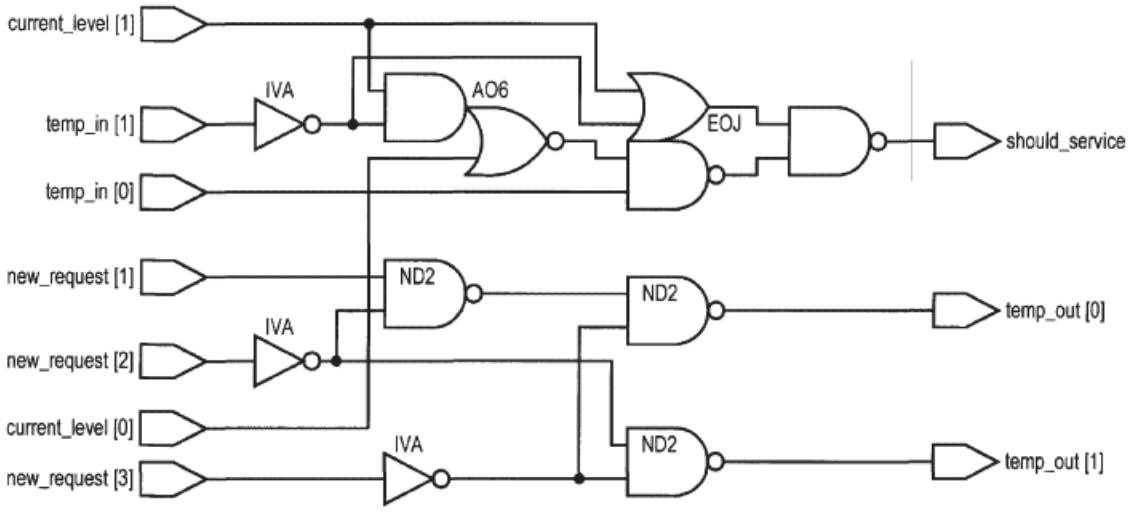


Figure 18, above, illustrates the circuit obtained by optimizing the circuit of Figure 17—a circuit generated by translating the VHDL source code of Figure 16. *Id.* at col. 10, ll. 9-12.

Mentor Graphics argues that Gregory fails to disclose the claimed "instrumentation signal" because "temp_out" does not involve additional logic, but instead simply preserves identified circuit nets during the optimization process. PO Resp. 53-56. This argument, however, requires a narrower construction of the term "instruction signal" than we have adopted, as discussed above.

Mentor Graphics also argues, without detailed explanation, that "[t]he proposed challenges to claims 3, 4, 5, 6, 7, 9, 11, 28, and 29 rely on portions of Gregory that are not arranged as recited in the claim." *Id.* at 40. This argument appears to be based on the statement in Gregory that Figures 12-19 "show another way to use probes to evaluate the performance of blocks of HDL code." Ex. 2007, col. 13, ll. 29-31. Thus, according to Mentor Graphics, the challenges to all the recited claims improperly combine portions of Gregory concerning Figures 12 and 16 with an alternate embodiment described in Figures 8 and 9. We are not persuaded that

Figures 12 through 19 are an entirely separate embodiment from that described in Figures 8 and 9. Instead, the difference between the two examples is that the initial source code differs. *See id.* at Figs. 8, 12. Furthermore, as described below, Gregory discloses all the limitations of claim 5 without any improper combination of embodiments.

We conclude that Gregory discloses all the limitations of claim 5 arranged as recited in the claim. Gregory discloses "a method of generating a gate level design." Ex. 1007, Abstract. Gregory also discloses "synthesizing the source code into a gate-level design having the instrumentation signal" by describing that the translation, or synthesis process, injects information into the resulting netlist when a probe is encountered, which results in components of the final circuit that are traceably related to the source code. Ex. 1007, col. 8, ll. 21-30. Either "tempout," as shown in Figure 9, or "temp_out," as shown in Figure 18, qualifies as "instrumentation signals" as construed above.

Finally, we determine that Gregory discloses "creating an instrumentation signal associated with at least one synthesizable statement contained in a register transfer level (RTL) synthesizable source code" as required by claim 5. For example, Gregory states that "any analytic result related to [tempout of Figure 9] . . . can be identified with the probe statement 401 in the HDL." Ex. 1007, col. 13, ll. 15-20. This language describes an "instrumentation signal" ("tempout") "associated" (identified) "with at least one synthesizable statement" (statement 401 in the HDL of Figure 8).

4. Dependent claims 6 and 7

Claim 6 depends from claim 5 and adds the limitation:

> wherein step a) further comprises the step of:
>
> (i) inserting a unique variable assignment statement into the source code, wherein the variable assignment statement is adjacent to at least one associated sequential statement; and
>
> (ii) inserting a unique output signal assignment statement into the source code, wherein the unique output signal is assigned a value associated with the unique variable.

Synopsys relies on Figures 16 and 18 of Gregory as disclosing this additional limitation. Pet. 35-36. Specifically, Synopsys asserts that the statements in Figure 16 adjacent each of the if, elsif, and else statements ("new_level <=") qualify as unique local variable statements. *Id.* at 35. Synopsys adds that synthesizing the VHDL code in Figure 16 produces the circuit of Figure 18 with "instrumentation signals" "temp_out" that are "assigned the value of the new_level local variable with a[n] initial, second value." *Id.* at 36.

Mentor Graphics argues that Gregory does not disclose the "variable assignment statement" limitation of claim 6. PO Resp. 56-57. Again, we give substantial weight to Mentor Graphics's expert testimony on this issue. Dr. Sarrafzadeh testifies, persuasively, that the "new_level" assignment in Figure 16 is a *signal* assignment and not a *variable* assignment. Ex. 2027 ¶ 84 (explaining that in VHDL variables are declared by the term "VARIABLE" and signals are assigned using the symbol ":="). Dr. Sarrafzadeh explains that "Gregory does not disclose including variable assignments, but instead only preserves signal assignments," which are substantively different than variable assignments. *Id.* at ¶¶ 85-87.

In response, Synopsys argues that Mentor Graphics does not use the broadest reasonable construction of the term "variable assignment statement" and improperly limits the scope of the claim to distinctions in terms provided by VHDL when the specification broadly talks about other HDL languages. Reply 13-14. We are not persuaded by Synopsys's argument. Mentor Graphics, in explaining its argument in terms of VHDL, is not limiting the scope of the claimed subject matter, but instead is responding to Synopsys's assertion that certain VHDL code shown in Gregory discloses the limitations of claim 6. We conclude that Synopsys has not met its burden with regard to claim 6. For the same reasons, we conclude that Synopsys has not met its burden with regard to claim 7, which depends from claim 6.

5. Dependent Claims 8 and 9

Claim 8 depends from claim 5 and adds the limitation:

> wherein step a) is repeated to create a unique instrumented output signal for each list of sequential statements in the source code, wherein each list corresponds to a synthesizable executable branch of the source code.

Synopsys relies on Figure 16 of Gregory as disclosing this additional limitation. Pet. 36-37. Specifically, Synopsys asserts that the "VHDL of FIG. 16 creates a unique instrumented output signal corresponding to the unique values of new_level for each list of sequential (if-then) statement," each of the statements being in a different branch of the VHDL. *Id.* at 37.

Claim 9 depends from claim 5 and adds the limitation:

> c) generating cross-reference instrumentation data mapping each statement in a selected list to the instrumented output signal associated with that list for every list in the source code.

Synopsys relies on Figure 16 and related language in Gregory as disclosing this limitation. Pet. 37 (citing Ex. 1007, col. 8, ll. 35-41).

For claim 8, Mentor Graphics relies solely on its argument made for claim 5 that Gregory fails to disclose the claimed "instrumentation signal" because "temp_out" does not involve additional logic, but instead simply preserves identified circuit nets during the optimization process. PO Resp. 59. As discussed, this argument is not persuasive.

For claim 9, Mentor Graphics also argues, without detailed explanation, that "[t]he proposed challenges to claims 3, 4, 5, 6, 7, 9, 11, 28, and 29 rely on portions of Gregory that are not arranged as recited in the claim." *Id.* at 40. As discussed above, we do not find this argument persuasive.

We conclude that Gregory discloses all the limitations of dependent claims 8 and 9.

6. Dependent Claim 11

Claim 11 depends from claim 5. Mentor Graphics argues that in its claim chart, Synopsys does not assert that claim 11 is anticipated by Gregory, but only that it would have been obvious over Gregory. Pet. 38. In reply, Synopsys states that "the Petition explicitly states that 'Gregory (Ex. 1007) *anticipates* claims 1-9, 11-14, 24-25 and 28-33 under section 102.'" Reply 15 (citing Pet. 4, 12, 38). This statement, however, is not evidence of anticipation. We agree with Mentor Graphics that nothing in the petition or in further briefing points to any language in Gregory that discloses all the limitations of claim 11. Thus, Synopsys has not met its burden with regard to claim 11.

E. Mentor Graphics's Motion to Amend Claims

Mentor Graphics filed a motion to amend claims. Paper 31 ("Mot. to Amend"). Mentor Graphics proposes nine substitute claims 34-36 and 38-43.[3] Mentor Graphics proposes that substitute claims 34, 35, and 36 are contingent substitute claims to replace original independent claims 1, 5, and 28, respectively, to be considered only if the original patent claim it replaces is unpatentable. *Id.* at 5. Similarly, proposed substitute claims 38-43 are contingent substitute claims to replace original dependent claims 3, 6, 8, 9, 11, and 29, respectively. *Id.* Because we determine that claims 5, 8, and 9 are anticipated by Gregory, the contingency has materialized for these claims, and, thus, we consider proposed substitute claims 35, 40, and 41.

As the moving party, Mentor Graphics bears the burden of proof to establish that it is entitled to the relief requested. 37 C.F.R. § 42.20(c). The proposed amendment is not entered automatically, but only upon Mentor Graphics's having demonstrated the patentability of those substitute claims.

Mentor Graphics contends that the proposed substitute independent claim, 35, "introduce[s] language to more explicitly recite the meaning of the term 'instrumentation signal'" and that the dependent claims are changed solely to change the dependency to the proposed substitute independent claim that corresponds to the dependent claim's respective independent base claim. Mot. to Amend 5. Proposed substitute independent claim 35 is reproduced below, with underlined text indicating material inserted relative to that claim:

[3] Mentor Graphics initially proposed substitute claim 37 replace original claim 2 without any contingency. Mot. to Amend 5. Subsequently, this request was withdrawn. Paper 39, 1.

> 35. A method of generating a gate level design, comprising the steps of:
>
> a) creating an instrumentation signal associated with at least one synthesizable statement contained in a register transfer level (RTL) synthesizable source code; and
>
> b) synthesizing the source code into a gate-level design having the instrumentation signal, the synthesizing comprising generating instrumentation logic to provide the instrumentation signal, the instrumentation logic comprising instrumentation logic circuitry that is additional to circuitry specified in the source code.

Proposed dependent claims 40 and 41 are identical to the claims they are to replace, claims 8 and 9, except that instead of depending from claim 5, proposed claims 40 and 41 would depend from proposed claim 35. Mot to Amend 4.

1. No Broadening of Scope

Proposed substitute claims may not enlarge the scope of original patent claims. 35 U.S.C. § 316(d)(3); 37 C.F.R. § 42.121(a)(2)(ii). Proposed substitute claims 35, 40, and 41 merely add features to the claims for which they substitute and do not remove any limitation therefrom. Accordingly, no issue exists with regard to the prohibition against broadening original patent claims.

2. Patentability over Prior Art

a. Anticipation by Gregory

Mentor Graphics asserts that the technique disclosed in Gregory prevents already synthesized circuit elements and signals from being altered structurally during optimization. Mot. to Amend 13 (citing Ex. 2027 ¶¶ 49-52). According to Mentor Graphics, in Gregory, "[n]o additional logic is added to the relabeled signals such that they are structurally changed in any

way to indicate that the signals have been instrumented." *Id.* at 13-14 (citing Ex. 2027 ¶¶ 63, 67-69).

Synopsys argues that Gregory's Figure 18 discloses the additional instrumentation logic circuitry limitation required by the proposed substitute claims. Opp. 4-6. Synopsys's expert, Dr. Brad Hutchings, testifies that the addition of logic gates to Figure 18 is apparent by comparison with Figure 14. Ex. 1013 ¶ 31. Figure 14 of Gregory, reproduced below, shows the circuit that results from optimizing the circuit shown in Figure 13, which in turn is the circuit resulting from translating the VHDL source code shown in Figure 12. Ex. 1007, col. 14, ll. 5-7.

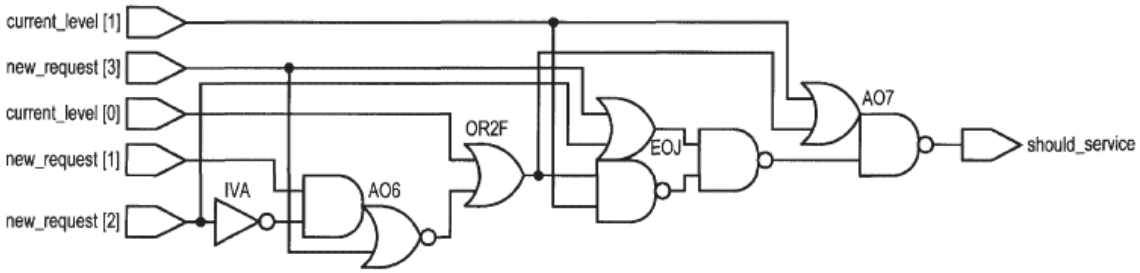


Figure 14, above, is a schematic for the circuit resulting from optimizing a circuit created by translating VHDL source code. *Id.* Figure 18, reproduced below, shows the circuit that results from optimizing the circuit shown in Figure 17, which in turn is the circuit resulting from translating the VHDL source code shown in Figure 16. *Id.* at col. 10, ll. 9-12.

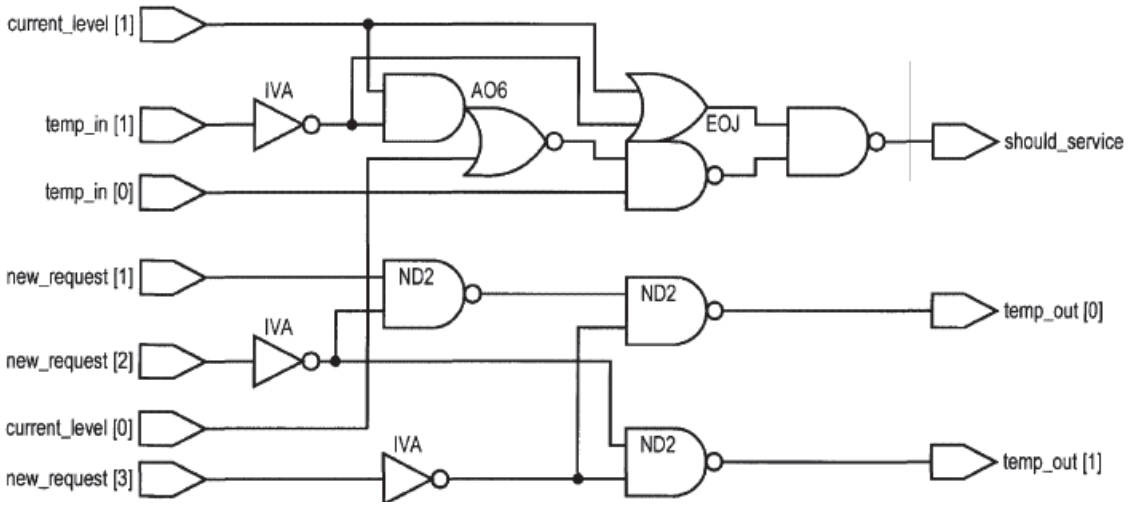


Figure 18, above, is a schematic for the circuit resulting from optimizing a circuit created by translating VHDL source code. *Id.* at col. 14, ll. 30-36. Because the VHDL source code in Figure 16 is the same as the VHDL source code in Figure 12 with block probes, or instrumentation, Figure 14 shows the circuit that results from translating uninstrumented source code, and Figure 18 shows the circuit that results from translating instrumented source code. Ex. 1013 ¶ 31; Ex. 1007, Fig. 12, 16; col. 9, ll. 64-65 ("Fig. 12. VHDL source without probes using two process blocks."); col. 10, l. 7 ("Fig. 16. VHDL source with two block probes installed.").

Dr. Hutchings testifies that the circuits of Figures 14 and 18 are "functionally identical but structurally different" and testifies that Figure 18 has a higher gate count than Figure 14. Ex. 1013 ¶ 32. Dr. Hutchings, however, does not testify affirmatively that the structural differences or the additional gate count are the result of the probes, as required by the proposed amendment, as opposed to the optimization process. *See id.* at ¶¶ 31-36.

Mentor Graphics responds that the differences in logic discussed by Dr. Hutchings come from the process of optimization, not additional logic resulting from probe statements. Paper 39, 2 ("Reply Mot. to Amend")

(citing Ex. 2027 ¶¶ 44, 51-53, 61-63). We are persuaded by Mentor Graphics's arguments and evidence that Gregory does not show "instrumentation logic comprising instrumentation logic circuitry that is additional to circuitry specified in the source code," as required by proposed substitute claims 35, 40, and 41.

b. Obviousness over Gregory

Mentor Graphics asserts that "one skilled in the art would not have been motivated to modify *Gregory* to include instrumentation logic circuitry that is additional to circuitry specified in the source code" because Gregory concerns software simulation and, therefore, eliminates the need for additional or generated instrumentation logic. Mot. to Amend 13-14 (citing Ex. 2027 ¶¶ 21, 27). Mentor Graphics asserts that "in view of the fundamental technical differences between Gregory and the claimed subject matter, any modification to Gregory such that the technique involved generating instrumentation logic circuitry that is additional to circuitry specified in the source code would change the principle of operation of *Gregory*." *Id.* at 14.

Synopsys argues that Mentor Graphics does not establish the nonobviousness of the proposed substitute claims over Gregory because it does not address the basic skill set possessed by a person of ordinary skill in the art and points to no evidence to support the assertion that one skilled in the art would not have been motivated to modify Gregory to include the additional logic circuitry limitation. Opp. 6. Dr. Hutchings testifies that "[n]umerous engineering texts and technical literature available to a person of ordinary skill in the art at the time of the invention taught inserting additional logic to debug circuits" and that an ordinary skilled artisan would

have known to use similar techniques when debugging synthesizable HDL. Ex. 1013 ¶¶ 37-38 (citing Ex. 1014-18). We give substantial weight to Dr. Hutchings's testimony on this issue, which is based on support from objective sources.

Mentor Graphics responds that Synopsys's "failure to present a strong showing of obviousness based on other references . . . render the alleged evidence insufficient to demonstrate obviousness." Reply Mot. to Amend 5. Mentor Graphics, however, misstates the burden required in this situation. It is Mentor Graphics who has the burden to show that it is entitled to the proposed substitute claims because they are patentable. Synopsys does not bear the burden to show that the claims are unpatentable.

We conclude, based on the record, that Mentor Graphics has not met its burden to show that independent claim 35 or claims 40 and 41, which depend from claim 35, would not have been obvious to a person of ordinary skill in the art based on the disclosure of Gregory.

c. Mentor Graphics's Burden

Moreover, distinguishing the proposed substitute claims only from the prior art references applied to the original patent claims is insufficient to demonstrate general patentability over prior art. As the moving party, a patent owner bears the burden to show entitlement to the relief requested. 37 C.F.R. § 42.20(c).

Mentor Graphics makes the conclusory statement, unsupported by evidence, that "the Patent Owner believes *Gregory* to be the closest known prior art and therefore believes the proposed substitute claims to be patentable over all known prior art." Mot. to Amend 14-15; *see also id.* at 11 ("Patent Owner believes *Gregory* to be the closest known prior art and is

not currently aware of any other prior art that would affect the patentability of the substitute claims.").

This statement is insufficient, without discussing the level of ordinary skill in the art, and what was previously known, with respect to each added feature, including the ordinary skill set possessed by such a hypothetical person. For each proposed claim, Mentor Graphics focuses on the added feature requiring additional instrumentation logic circuitry. However, Mentor Graphics reveals little, if anything, about the level of ordinary skill and what was previously known with respect to that feature.

In the context of the claim element added by Mentor Graphics, it is essential to know whether synthesizing source code, including additional instrumentation logic circuitry, pre-existed the claimed invention, in any context, and, if so, how it worked. Otherwise, Mentor Graphics is expected, reasonably, to explain such pre-existing art, and why it would not have been applicable to render the invention of the proposed substitute claims obvious to one with ordinary skill in the art. Mentor Graphics has failed to do either.

Without having discussed sufficiently, in its motion, the level of ordinary skill in the art and what was previously known regarding the features on which Mentor Graphics focuses for establishing patentability, Mentor Graphics has not, in its motion, set forth a prima facie case for the relief requested—that independent claim 35 and claims 40 and 41, are patentable—or satisfied its burden of proof.

3. Written Description Support

Because Mentor Graphics has not shown patentability of the proposed substitute claims over the prior art, we do not reach whether it has shown that the proposed substitute claims have written description support in the

'584 application as filed. We note that Mentor Graphics should have cited to the disclosure of the '584 application as filed rather than the disclosure of the '376 patent as issued.

Mentor Graphics's Motion to Amend Claims is *denied*.

F. Mentor Graphics's Motion to Exclude Evidence

Mentor Graphics filed a Motion to Exclude Evidence (Paper 42) seeking to exclude the Declaration testimony of Dr. Hutchings because it is not competent expert testimony. Mentor Graphics argues that Dr. Hutchings did not have an understanding of the claimed subject matter as a whole, including each limitation of the claim, and therefore his testimony fails to satisfy the criteria of Federal Rule of Evidence 702. Paper 42, 3-8. Mentor Graphics bases this assertion on Dr. Hutchings's testimony that he had not formed an opinion on the meaning of "execution status" or "instrumentation signal," but instead assumed that "the Board already ruled on anticipation, so I focused on the amended language" of the proposed substitute claims. *Id.* at 7-8 (quoting Ex. 2032, 97:3-11).

We agree with Synopsys that Dr. Hutchings's testimony should not be excluded. Mentor Graphics's objections to Dr. Hutchings's testimony go to the weight and sufficiency of his testimonial evidence, rather than its admissibility. *See Liquid Dynamics Corp. v. Vaughan Co.*, 449 F.3d 1209, 1221 (Fed. Cir. 2006) (citing *Quiet Tech. DC-8, Inc. v. Hurel-Dubois UK Ltd.*, 326 F.3d 1333, 1344-45 (11th Cir. 2003)); *In re TMI Litig.*, 193 F.3d 613, 692 (3d Cir. 1999) ("So long as the expert's testimony rests upon 'good grounds,' it should be tested by the adversary process—competing expert testimony and active cross-examination." (quoting *Ruiz–Troche v. Pepsi Cola of Puerto Rico Bottling Co.*, 161 F.3d 77, 85 (1st Cir.1998))));

Wilmington v. J.I. Case Co., 793 F.2d 909, 920 (8th Cir.1986) ("Virtually all the inadequacies in the expert's testimony urged here by [the defendant] were brought out forcefully at trial . . . . These matters go to the weight of the expert's testimony rather than to its admissibility."). Mentor Graphics had the opportunity to address any alleged deficiencies in Dr. Hutchings's testimony in the Reply in support of the Motion to Amend. *See* Reply Mot. to Amend 1, 3 (stating that Dr. Hutchings's testimony is "incompetent and entitled to no weight").

Nevertheless, we have reviewed the testimony in question and conclude that Dr. Hutchings has demonstrated appropriate credentials, adequate preparation, and sufficient understanding of the '376 patent. *See, e.g.*, Ex. 1013 ¶¶ 4-11, 19-22. Mentor Graphics does not point to any authority supporting its position. Here, Dr. Hutchings presumed the Board would adopt certain constructions for the terms in the claims and stated his opinions based on those constructions. This presumption was logical given that Dr. Hutchings opines only on proposed substitute claims that are contingent on a finding of anticipation. Nothing about this presumption indicates a lack of understanding of the claims.

Mentor Graphics's Motion to Exclude Evidence is *denied.*

G. Synopsys's Motion to Exclude Evidence

Synopsys filed a Motion to Exclude Evidence (Paper 44) seeking to exclude (1) all of Mentor Graphics's exhibits relating to assignor estoppel because they are not relevant; (2) all of Mentor Graphics's exhibits relating to the post-2006 relationship between EVE and Synopsys because they are not relevant; and (3) Exhibits 2030 and 2031 because they were not cited or explained in any paper, and Exhibit 2033 because it relates to conception

date of the '376 patent, which is not at issue in this proceeding.

We find it unnecessary to consider the specific objections to the admissibility of exhibits relating to assignor estoppel because Mentor Graphics has failed to demonstrate that assignor estoppel provides an exception to the statutory mandate, even assuming those exhibits to be admissible.

Similarly, we find it unnecessary to consider the specific objections to the admissibility of exhibits relating to the post-2006 relationship between EVE and Synopsys, because Mentor Graphics has failed to demonstrate that a real party-in-interest or privity relationship between EVE and Synopsis existed during the relevant time, even assuming those exhibits to be admissible.

Finally, we find it unnecessary to consider the objections to the admissibility of Exhibits 2030, 2031, and 2033. As pointed out by Synopsys, Exhibits 2030 and 2031 were not cited in any of the briefs. Mentor Graphics explains that the Exhibits are relevant to Dr. Hutchings's retraction of paragraph 33 of his deposition. Paper 47, 6. However, because Dr. Hutchings retracted paragraph 33, we did not rely on this portion of his testimony in making our decision. Thus, we also have not relied on Exhibits 2030 and 2031. We similarly have not relied on Exhibit 2033, a declaration submitted by Mentor Graphics to swear behind the Boubezari reference (*Id.* at 7) because we did not reach that issue in deciding this case.

The motion is dismissed as moot, because even considering the evidence that Synopsys seeks to exclude, we either have not reached the issue to which the evidence relates or we have decided the issue in Synopsys's favor.

III. CONCLUSION

Synopsys has not shown by a preponderance of the evidence that claims 1-4, 6, 7, 11, 28, and 29 of the '376 patent are unpatentable under 35 U.S.C. § 102(e) over Gregory.

Synopsys has shown by a preponderance of the evidence that claims 5, 8, and 9 of the '376 patent are unpatentable under 35 U.S.C. § 102(e) over Gregory.

Mentor Graphics has not shown that its proposed substitute claims 34-36 and 38-43 are patentable over the prior art.

Accordingly, it is

ORDERED that claims 5, 8, and 9 of the '376 patent are CANCELLED;

FURTHER ORDERED that Mentor Graphics's Motion to Amend Claims is *denied*;

FURTHER ORDERED that Mentor Graphics's Motion to Exclude Evidence is *denied*;

FURTHER ORDERED that Synopsys's Motion to Exclude Evidence is *dismissed.*

PETITIONER:

William H. Wright
Travis Jensen
Orrick, Herrington & Sutcliffe, LLP
Email: wwright@orrick.com
Email: tjensen@orrick.com

PATENT OWNER:

Christopher L. McKee
Michael S. Cuviello
Banner & Witcoff, Ltd.
Email: mentoripr@bannerwitcoff.com

Mark Miller
O'Melveny & Myers LLP
Email: markmiller@omm.com

U.S. Patent No. 6,240,376

Dated May 29, 2001

US006240376B1

(12) **United States Patent**
Raynaud et al.

(10) **Patent No.: US 6,240,376 B1**
(45) **Date of Patent: May 29, 2001**

(54) **METHOD AND APPARATUS FOR GATE-LEVEL SIMULATION OF SYNTHESIZED REGISTER TRANSFER LEVEL DESIGNS WITH SOURCE-LEVEL DEBUGGING**

(75) Inventors: **Alain Raynaud**, Paris; **Luc M. Burgun**, Creteil, both of (FR)

(73) Assignee: **Mentor Graphics Corporation**, Wilsonville, OR (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/127,584**

(22) Filed: **Jul. 31, 1998**

Related U.S. Application Data

(63) Continuation-in-part of application No. 09/122,493, filed on Jul. 24, 1998.

(51) **Int. Cl.**[7] **G06F 17/50**
(52) **U.S. Cl.** **703/15**; 703/14; 714/741
(58) **Field of Search** 395/500.35, 500.36, 395/500.37; 714/724, 734; 703/14, 15, 16; 716/4, 724, 734, 741

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,220,512		6/1993	Watkins et al.	364/489
5,253,255	*	10/1993	Carbine	714/734

(List continued on next page.)

OTHER PUBLICATIONS

Chen et al., "A Source–Level Dynamic Analysis Methodology and Tool for High–LevelSynthesis", Proceedings of the Tenth International Symposium on System Synthesis, 1997, pp. 134–140, Sep. 1997.*

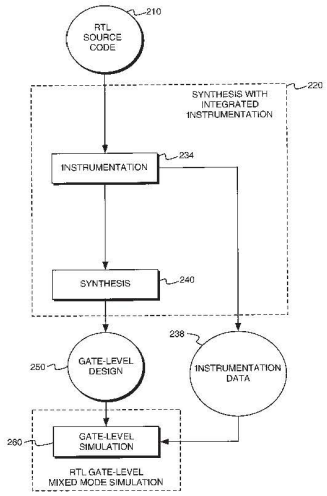
(List continued on next page.)

Primary Examiner—Kevin J. Teska
Assistant Examiner—Douglas W. Sergent
(74) *Attorney, Agent, or Firm*—Columbia IP Law Group, LLC

(57) **ABSTRACT**

Methods of instrumenting synthesizable source code to enable debugging support akin to high-level language programming environments for gate-level simulation are provided. One method of facilitating gate level simulation includes generating cross-reference instrumentation data including instrumentation logic indicative of an execution status of at least one synthesizable register transfer level (RTL) source code statement. A gate-level netlist is synthesized from the source code. Evaluation of the instrumentation logic during simulation of the gate-level netlist facilitates simulation by indicating the execution status of a corresponding source code statement. One method results in a modified gatelevel netlist to generate instrumentation signals corresponding to synthesizable statements within the source code. This may be accomplished by modifying the source code or by generating the modified gate-level netlist as if the source code was modified during synthesis. Alternatively, cross-reference instrumentation data including instrumentation logic can be generated without modifying the gate-level design. The instrumentation logic indicates the execution status of a corresponding cross-referenced synthesizable statement. An execution count of a cross-referenced synthesizable statement can be incremented when the corresponding instrumentation signals indicates the statement is active to determine source code coverage. Source code statements can be highlighted when active for visually tracing execution paths. For breakpoint simulation, a breakpoint can be set at a selected source code statement. The corresponding instrumentation logic from the cross-reference instrumentation data is implemented as a simulation breakpoint. The simulation is halted at a simulation cycle where the values of the instrumentation signals indicate that the source code statement is active.

33 Claims, 22 Drawing Sheets



U.S. PATENT DOCUMENTS

5,325,309		6/1994	Halaviati et al.	364/488
5,423,023		6/1995	Batch et al.	395/500
5,461,576		10/1995	Tsay et al.	364/490
5,513,123	*	4/1996	Dey et al.	716/4
5,519,627		5/1996	Mahamood et al.	364/488
5,541,849		7/1996	Rostoker et al.	364/489
5,544,067		8/1996	Rostoker et al.	364/489
5,553,002		9/1996	Dangelo et al.	364/489
5,555,201		9/1996	Dangelo et al.	364/489
5,568,396		10/1996	Bamji et al.	364/491
5,598,344		1/1997	Dangelo et al.	364/489
5,615,356		3/1997	King et al.	395/500
5,623,418		4/1997	Rostoker et al.	364/489
5,632,032	*	5/1997	Ault et al.	709/100
5,727,187		3/1998	Lemche et al.	395/500
5,758,123		5/1998	Sano et al.	395/500
5,768,145		6/1998	Roethig	364/488
5,801,958		9/1998	Dangelo et al.	364/489
5,835,380		11/1998	Roethig	364/488
5,841,663		11/1998	Sharma et al.	364/490
5,870,308		2/1999	Dangelo et al.	364/489
5,870,585	*	2/1999	Stapleton	703/15
5,880,971	*	3/1999	Dangelo et al.	703/16
5,920,711		7/1999	Seawright et al.	395/500
5,937,190		8/1999	Gregory et al.	395/704
5,960,191		9/1999	Sample et al.	395/500.49
5,991,533		11/1999	Sano et al.	395/500.49
6,006,022		12/1999	Rhim et al.	395/500.02
6,009,256		12/1999	Tseng et al.	395/500.34

OTHER PUBLICATIONS

Kucukcakar et al., "Matisse: An Architectural Design Tool for Commodity IC's", IEEE Design & Test of Computers, vol. 15, Issue 2, pp. 22–33, Jun. 1998.*

Koch et al. "Debugging of Behavioral VHDL Specifications by Source Level Emulation", Proceedings of the European Design Automation Conference, pp. 256–261, Sep. 1995.*

Fang et al., "A Real–Time RTL Engineering–Change Method Supporting On–Line Debugging for Logic–Emulation Applications", Proceedings of the 34th Design Automation Conference, pp. 101–106, Jun. 1997.*

Howe, H., "Pre– and Postsynthesis Mismatches", IEEE International Conf. on Verilog HDL 1997, pp. 24–31, Apr. 1997.*

Postula et al., "A Comparison of High Level Synthesis and Register Transfer Design Techniques for Custom Computing Machines", Proc. of the 31st Hawaii Inter. Conf. on System Sciences, vol. 7, pp. 207–214, Jan. 1998.*

Orailoglu, A., "Microarchitectural Sythesis for Rapid BIST Testing", IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems, vol. 16, Issue 6, pp. 573–586, Jun. 1997.*

* cited by examiner

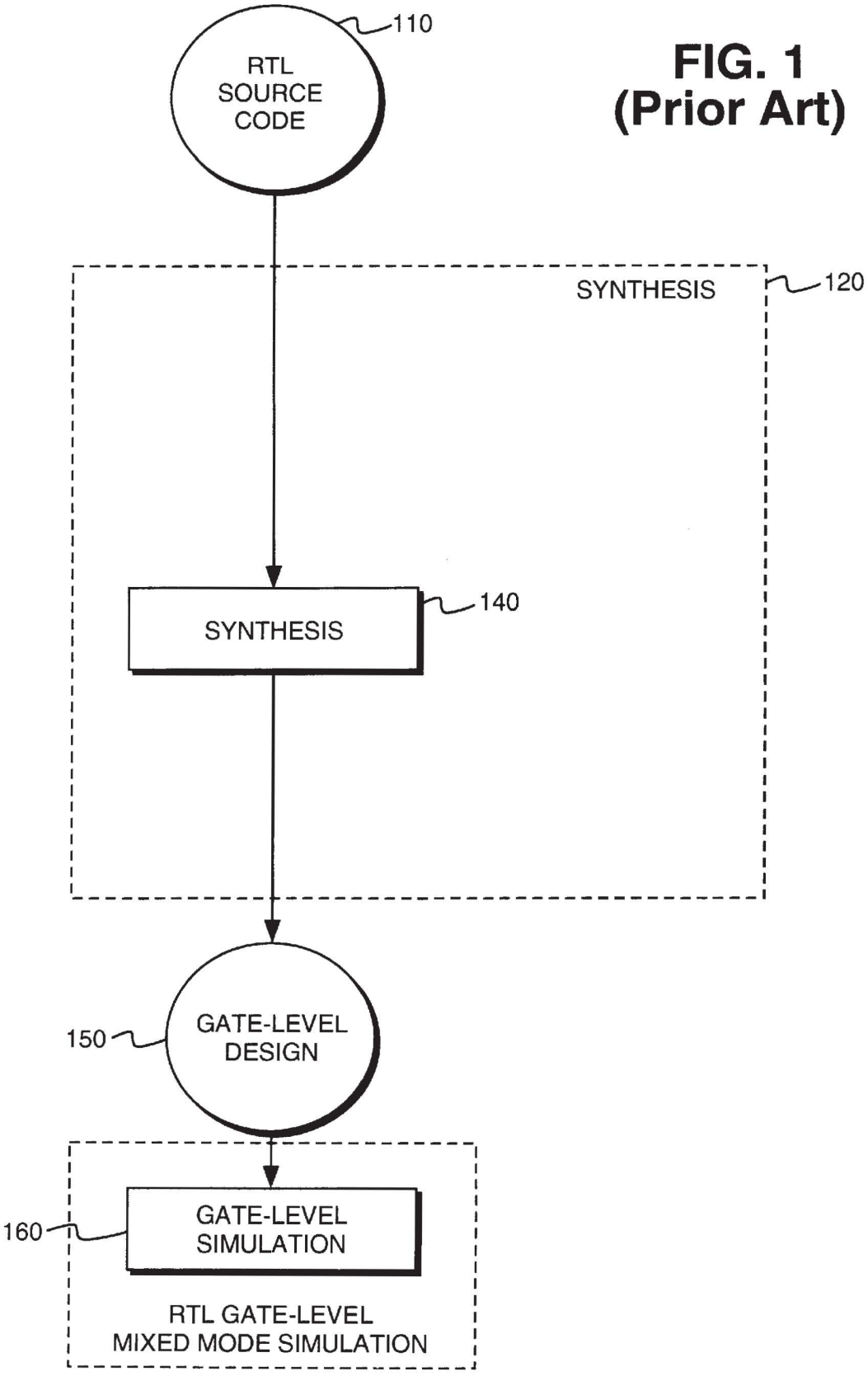


FIG. 1
(Prior Art)

FIG. 2

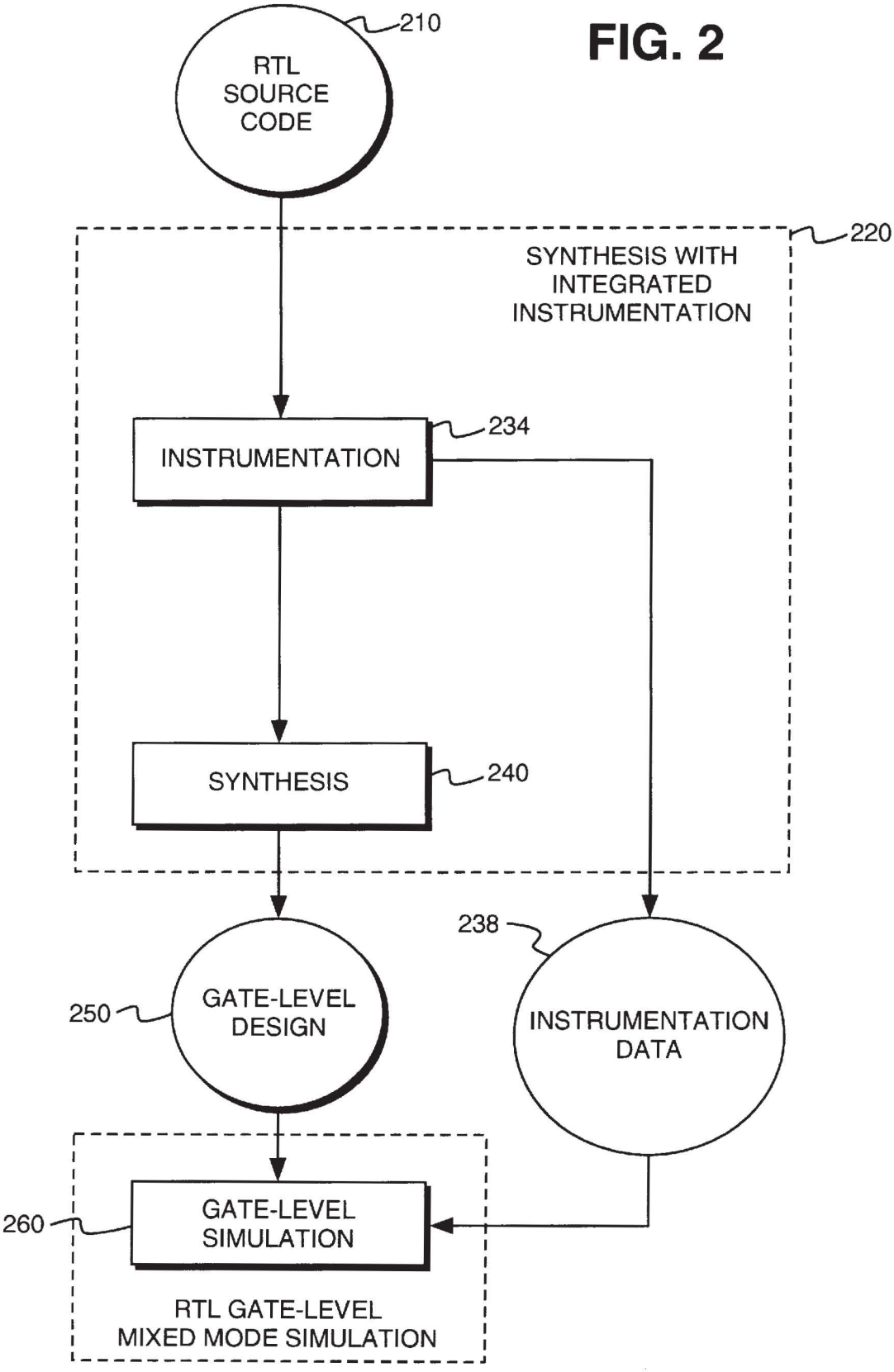

210
RTL SOURCE CODE
220
SYNTHESIS WITH INTEGRATED INSTRUMENTATION
234
INSTRUMENTATION
240
SYNTHESIS
238
250
GATE-LEVEL DESIGN
INSTRUMENTATION DATA
260
GATE-LEVEL SIMULATION
RTL GATE-LEVEL MIXED MODE SIMULATION

FIG. 3

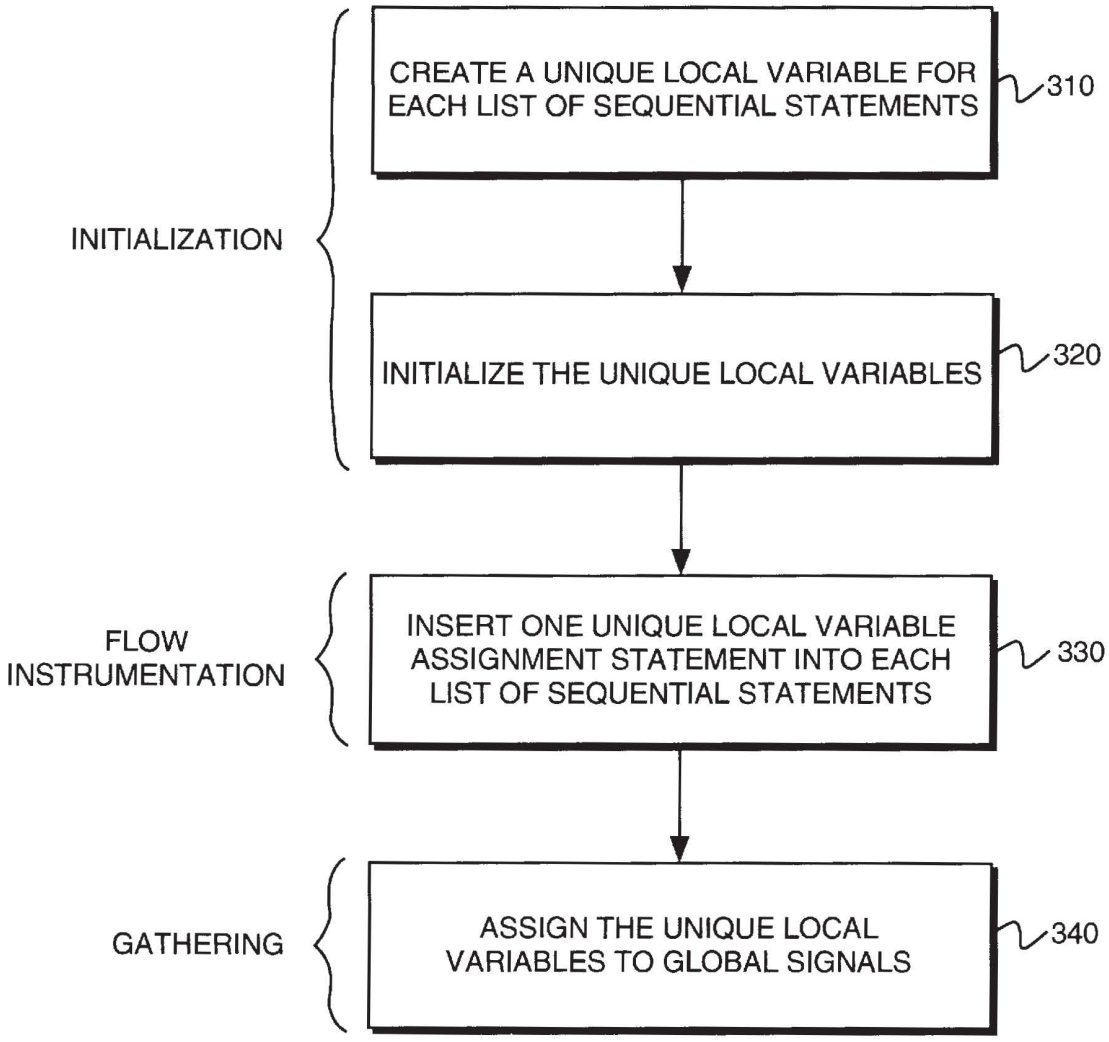

INITIALIZATION
CREATE A UNIQUE LOCAL VARIABLE FOR EACH LIST OF SEQUENTIAL STATEMENTS
310
INITIALIZE THE UNIQUE LOCAL VARIABLES
320
FLOW INSTRUMENTATION
INSERT ONE UNIQUE LOCAL VARIABLE ASSIGNMENT STATEMENT INTO EACH LIST OF SEQUENTIAL STATEMENTS
330
GATHERING
ASSIGN THE UNIQUE LOCAL VARIABLES TO GLOBAL SIGNALS
340

FIG. 4

400

```
ENTITY ALOOP IS

PORT(
        A : IN BIT_ VECTOR ( 0 TO 1 ) ;
        RESET : IN BOOLEAN;
        STATUS : OUT BOOLEAN ) ;

END ENTITY ALOOP ;


ARCHITECTURE RTL OF ALOOP IS

BEGIN

PROCESS ( A, RESET )

VARIABLE ZEROS, ONES : INTEGER ;

BEGIN
410 ---> IF ( RESET )                                   -- STATEMENT # 1
         THEN
420 --->     STATUS <= 0 ;                              -- STATEMENT # 2
         ELSE
430 --->     ZEROS := 0;                                -- STATEMENT # 3
440 --->     ONES := 0;                                 -- STATEMENT # 4
450 --->     FOR I IN 0 TO 1 LOOP                       -- STATEMENT # 5
460 --->         IF A ( I ) = '0'                       -- STATEMENT # 6
                 THEN
470 --->             ZEROS := ZEROS + 1 ;               -- STATEMENT # 7
                 ELSE
480 --->             ONES := ONES + 1 ;                 -- STATEMENT # 8
                 END IF ;
             END LOOP ;
490 --->     STATUS <= ( ZEROS > ONES ) ;               -- STATEMENT # 9
         END IF ;

END PROCESS ;
END ARCHITECTURE ;
```

FIG. 5

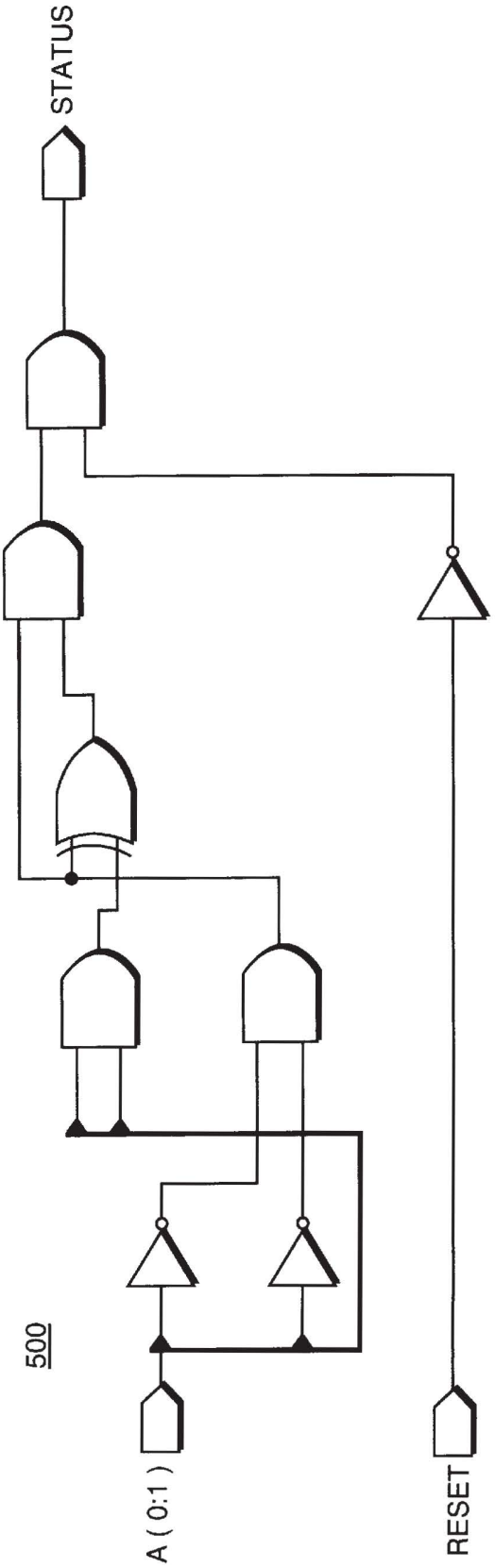
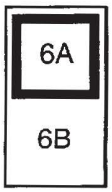


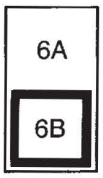
FIG. 6A



600

```
ENTITY ALOOP IS
PORT(
A : IN BIT_VECTOR (0 TO 1) ;
RESET : IN BOOLEAN ;
STATUS : OUT BOOLEAN ;
SIG_ TRACE1, SIG_ TRACE2, SIG_ TRACE3, SIG_ TRACE4, SIG_TRACE5,        } ~610
SIG_TRACE6 : OUT BIT
);
END ENTITY ALOOP ;

ARCHITECTURE RTL OF ALOOP IS
BEGIN
PROCESS (A, RESET)
VARIABLE TRACE1, TRACE2, TRACE3, TRACE4, TRACE5, TRACE6 : BIT ;  } ~612
VARIABLE ZEROS, ONES : INTEGER ;

BEGIN
TRACE1 :='0' ; TRACE2 :='0' ;
TRACE3 :='0' ; TRACE4 :='0' ;  } ~620
TRACE5 :='0' ; TRACE6 :='0' ;
```

FIG. 6B



600

```
630 ──► TRACE1 := '1' ;                  -- INSTRUMENT STATEMENT #1
        IF (RESET)                       -- STATEMENT #1
        THEN
632 ──►   TRACE2 := '1' ;                -- INSTRUMENT STATEMENT #2
          STATUS <= FALSE ;              -- STATEMENT #2
        ELSE
634 ──►   TRACE3 := '1' ;                -- INSTRUMENT STATEMENT #3, #4, #5, #9
          ZEROS := 0 ;                   -- STATEMENT #3
          ONES := 0 ;                    -- STATEMENT #4
          FOR I IN 0 TO 1 LOOP           -- STATEMENT #5
636 ──►     TRACE4 := '1' ;              -- INSTRUMENT STATEMENT #6
            IF A (I) ='0' ;              -- STATEMENT #6
            THEN
638 ──►       TRACE5 := '1' ;            -- INSTRUMENT STATEMENT #7
              ZEROS := ZEROS +1 ;        -- STATEMENT #7
            ELSE
640 ──►       TRACE6 := '1' ;            -- INSTRUMENT STATEMENT #8
              ONES := ONES +1;           -- STATEMENT  #8
            END IF;
          END LOOP;

642 ──► STATUS <= (ZEROS > ONES) ;       -- STATEMENT #9

        END IF ;
        SIG_ TRACE1 <= TRACE1 ; SIG_ TRACE2 <= TRACE2 ;  }
        SIG_ TRACE3 <= TRACE3 ; SIG_ TRACE4 <= TRACE4 ;  } ~ 650
        SIG_ TRACE5 <= TRACE5 ; SIG_ TRACE6 <= TRACE6 ;  }
        END PROCESS ;

        END ARCHITECTURE ;
```

FIG. 7

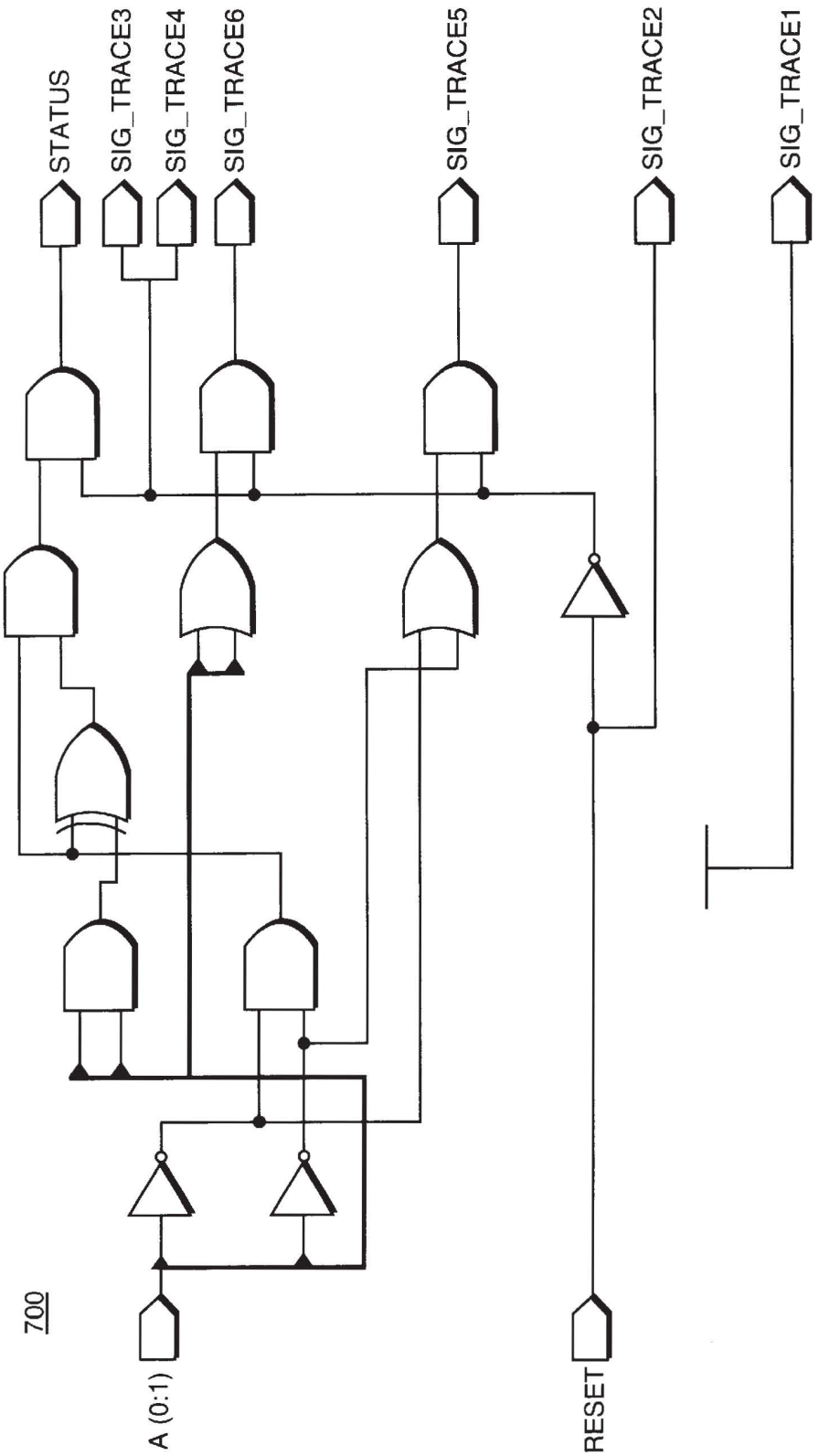

STATUS
SIG_TRACE3
SIG_TRACE4
SIG_TRACE6
SIG_TRACE5
SIG_TRACE2
SIG_TRACE1
700
A (0:1)
RESET

FIG. 8

800

```
MODULE SAMPLE (RESET, D, CLK, Q) ;

INPUT RESET ;
INPUT D ;
INPUT CLK ;
REG Q ;
OUTPUT Q ;


ALWAYS @ (CLK OR RESET OR D)
BEGIN
        IF (RESET==1)
            Q <= 0 ;
        ELSE
            IF (CLK==1)
                Q <= D ;

END


ENDMODULE
```

FIG. 9

900

```
MODULE SAMPLE
(RESET, D, CLK, Q, SIG_TRACE1, SIG_TRACE2, SIG_TRACE3, SIG_ TRACE4 ) ;

INPUT RESET ;
INPUT D ;
INPUT CLK ;
REG Q ;
OUTPUT Q ;

REG SIG_TRACE1, SIG_TRACE2, SIG_TRACE3, SIG_TRACE4 ;
OUTPUT SIG_TRACE1, SIG_TRACE2, SIG_TRACE3, SIG_TRACE4 ;

INTEGER TRACE1, TRACE2, TRACE3, TRACE4 ;

ALWAYS @ (CLK OR RESET OR D)
BEGIN
        TRACE1 = 0 ; TRACE2 = 0 ; TRACE3 = 0 ; TRACE4 = 0 ;
        TRACE1 = 1 ;
        IF (RESET==1)
        BEGIN
                TRACE2 = 1 ;
                Q <= 0 ;
        END
        ELSE
        BEGIN
                TRACE3 = 1 ;
                IF (CLK== 1)
                BEGIN
                        TRACE4 = 1 ;
                          Q <= D ;
                END
        END

SIG_TRACE1 = TRACE1 ;
SIG_TRACE2 = TRACE2 ;
SIG_TRACE3 = TRACE3 ;
SIG_TRACE4 = TRACE4 ;

END

ENDMODULE
```

FIG. 10

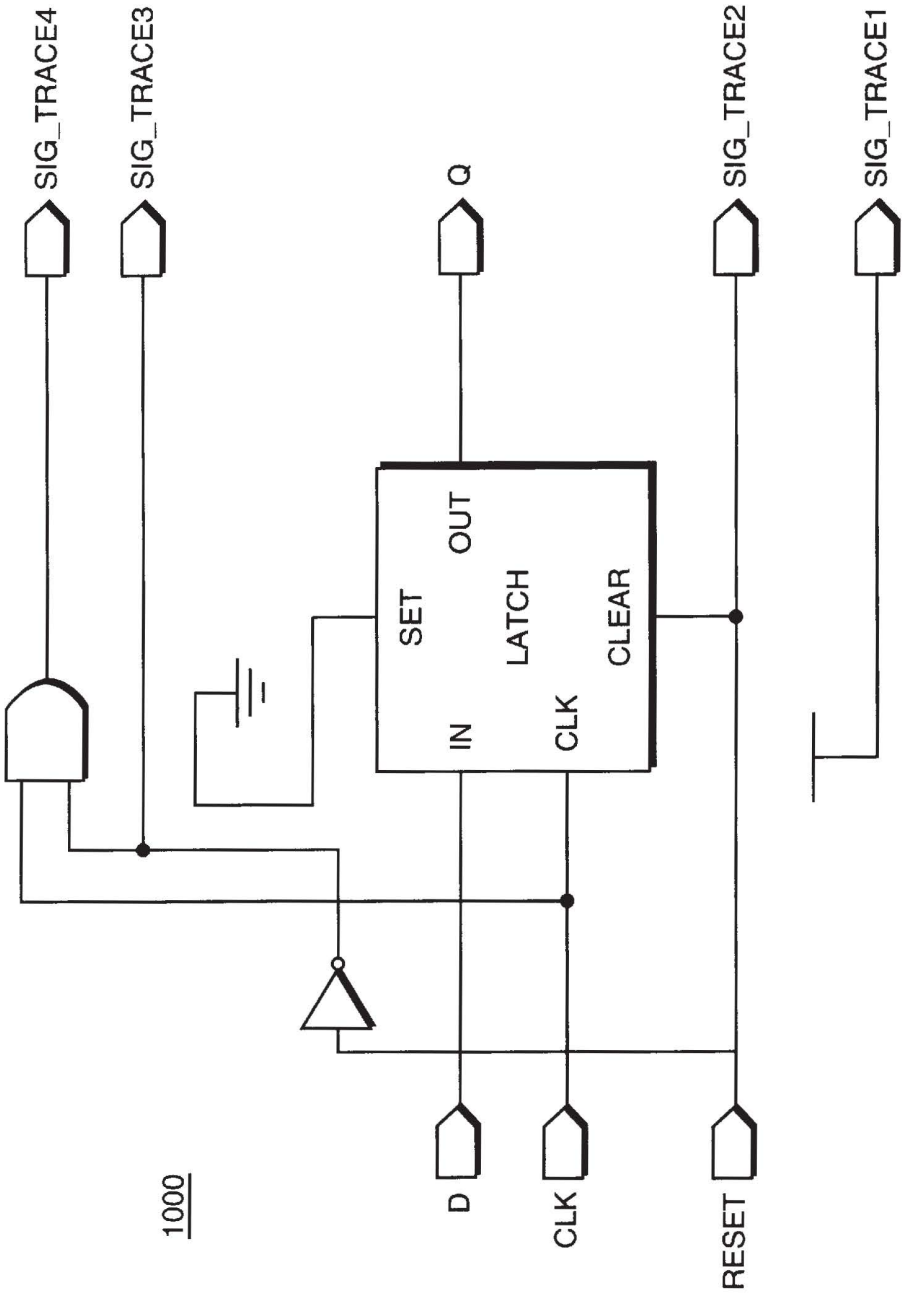

SIG_TRACE4
SIG_TRACE3
Q
SIG_TRACE2
SIG_TRACE1
SET
OUT
LATCH
CLEAR
IN
CLK
1000
D
CLK
RESET

FIG. 11

1100

```
PROCESS (CLK, D, RESET)
BEGIN

        IF (RESET = '1') THEN
                Q <= '0' ;

        ELSIF (CLK'EVENT AND CLK = '1') THEN
                Q <= D ;

        END IF;

END PROCESS
```

FIG. 12

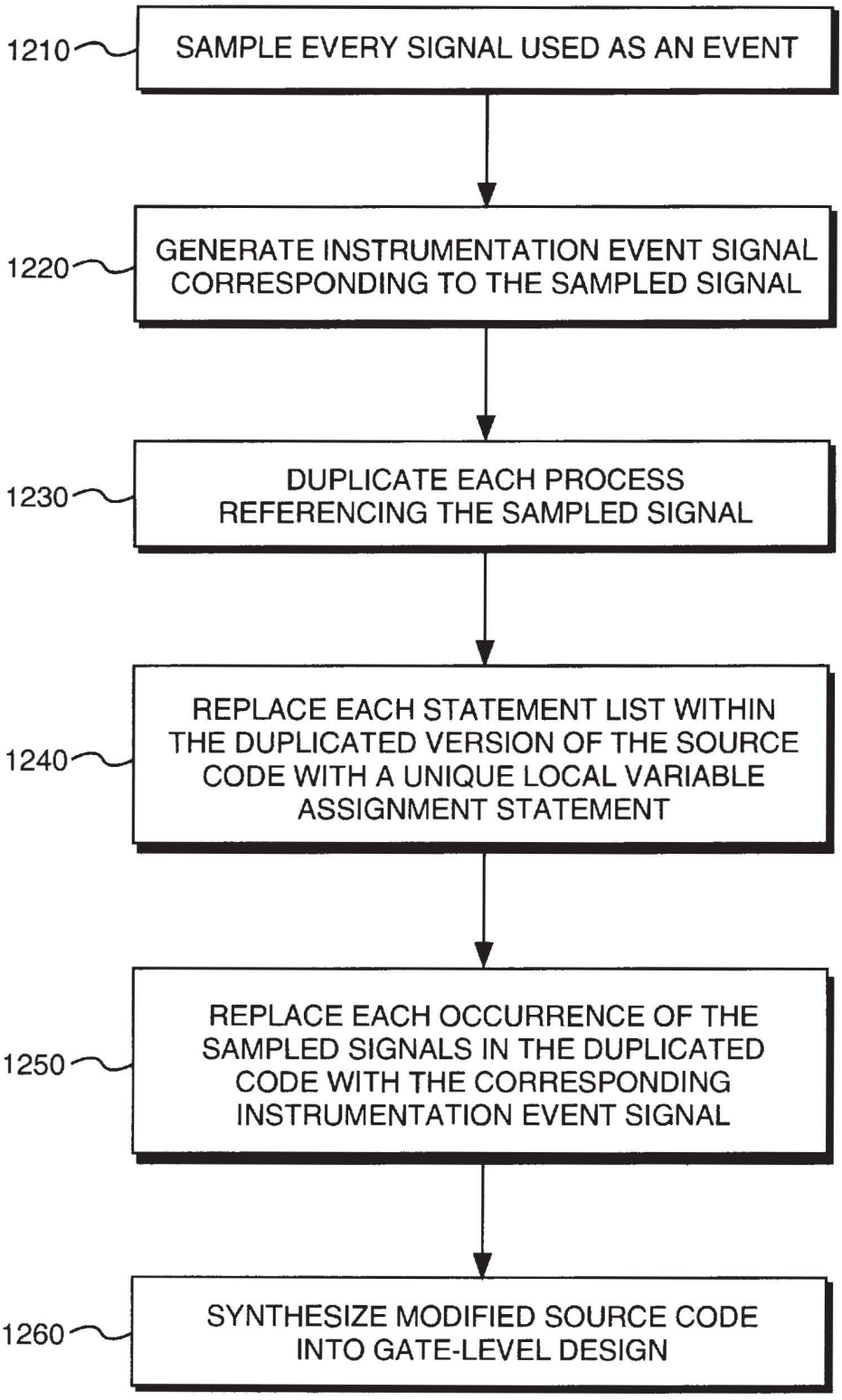

1210
SAMPLE EVERY SIGNAL USED AS AN EVENT
1220
GENERATE INSTRUMENTATION EVENT SIGNAL CORRESPONDING TO THE SAMPLED SIGNAL
1230
DUPLICATE EACH PROCESS REFERENCING THE SAMPLED SIGNAL
1240
REPLACE EACH STATEMENT LIST WITHIN THE DUPLICATED VERSION OF THE SOURCE CODE WITH A UNIQUE LOCAL VARIABLE ASSIGNMENT STATEMENT
1250
REPLACE EACH OCCURRENCE OF THE SAMPLED SIGNALS IN THE DUPLICATED CODE WITH THE CORRESPONDING INSTRUMENTATION EVENT SIGNAL
1260
SYNTHESIZE MODIFIED SOURCE CODE INTO GATE-LEVEL DESIGN

FIG. 13

1300

```
PROCESS (FAST_CLK)
BEGIN
        IF (FAST_CLK'EVENT AND FAST_CLK = '1')
        THEN
                SAMPLED_CLK <= CLK ;
        END IF
END PROCESS ;

CLK_EVENT <= SAMPLED_CLK / = CLK ;
CLK_STABLE <= SAMPLED_CLK = CLK ;
CLK_LASTVALUE <= SAMPLED_CLK ;
```

1310

```
PROCESS (CLK, D, RESET, CLK _ EVENT)
        VARIABLE TRACE1, TRACE2 : BIT ;
BEGIN
        TRACE1 := '0' ; TRACE2 := '0' ;
        IF (RESET = '1') THEN
                TRACE1 := '1' ;
        ELSIF (CLK_EVENT AND CLK = '1') THEN
                TRACE2 := '1' ;
        END IF;
SIG_TRACE1 <= TRACE1 ;  SIG_TRACE2 <= TRACE2 ;
END PROCESS ;
```

1320

```
PROCESS (CLK, D, RESET)
BEGIN
        IF (RESET = '1') THEN
                Q <= '0' ;
        ELSIF (CLK'EVENT AND CLK = '1') THEN
                Q <= D ;
        END IF ;
END PROCESS
```

1330

FIG. 14

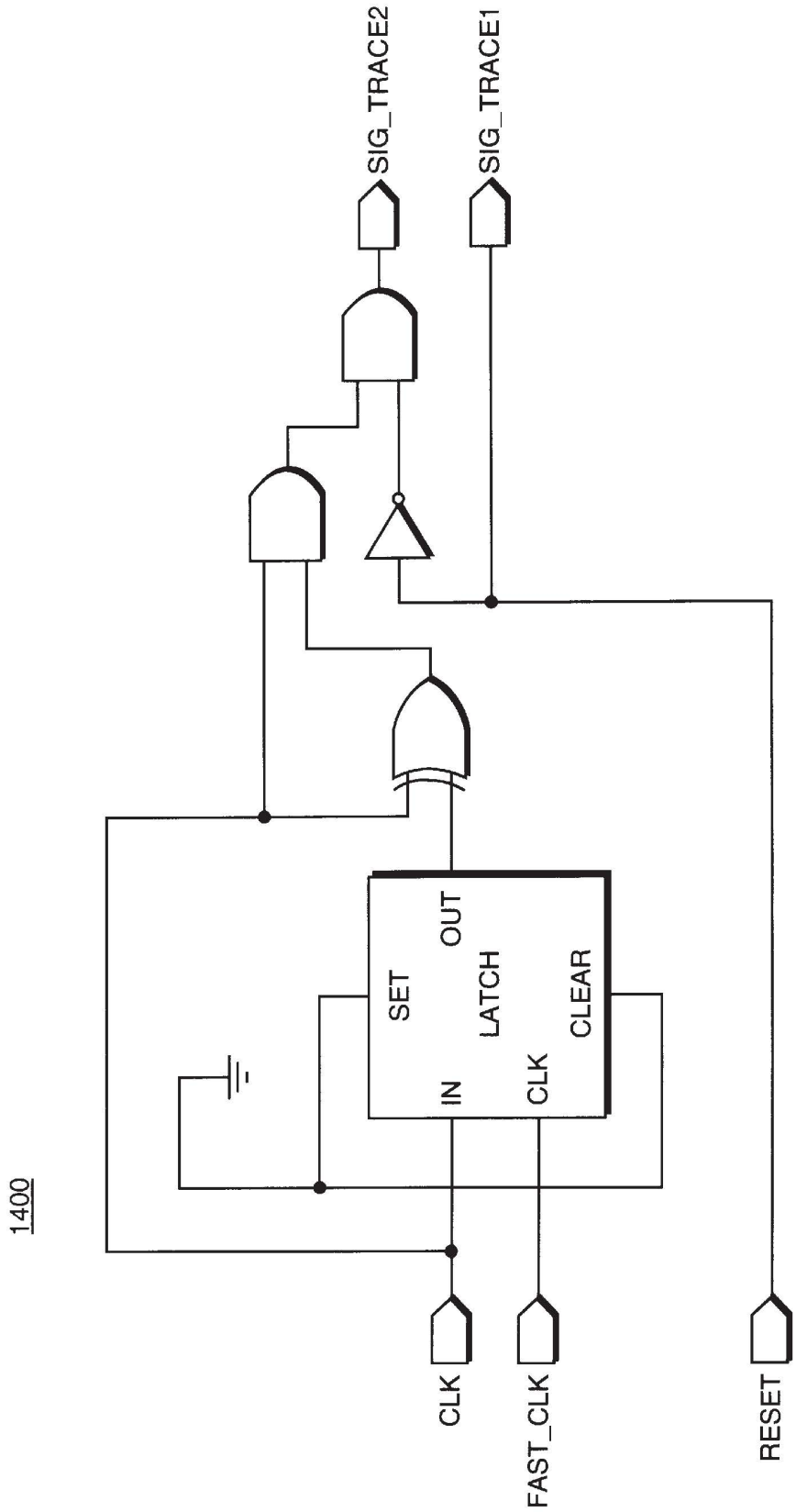

SIG_TRACE2
SIG_TRACE1
SET
OUT
LATCH
CLEAR
IN
CLK
1400
CLK
FAST_CLK
RESET

FIG. 15

1500

```
ALWAYS @ (POSEDGE CLK OR NEGEDGE RESET)
BEGIN
        IF (RESET == 0)
                Q <= 0 ;
        ELSE
                Q <= D ;
END
```

FIG. 16

1600

```
ALWAYS @ (POSEDGE FAST_CLK)
BEGIN
        SAMPLED_CLK <= CLK
        SAMPLED_RESET <= RESET ;
END
ASSIGN CLK_EDGE = SAMPLED_CLK ^ CLK ;
ASSIGN RESET_ EDGE = SAMPLED_ RESET ^ RESET;

INTEGER TRACE1, TRACE2;
REG [1:0] SIG_ TRACE ;
ALWAYS @ (CLK_EDGE OR RESET_EDGE OR CLK OR RESET)
BEGIN
        TRACE1 = 0 ; TRACE2 = 0 ;
        IF ((CLK_EDGE == 1) && (CLK == 1)II(RESET_EDGE == 1) && (RESET == 0))
                IF (RESET == 0)
                        TRACE1 =1;
                ELSE
                        TRACE2 =1;
                SIG_TRACE[0] = TRACE1 ;
                SIG_TRACE[1] = TRACE2 ;
END

ALWAYS @ (POSEDGE CLK OR NEGEDGE RESET)
BEGIN


        IF (RESET == 0)
            Q <=0 ;
         ELSE
            Q <= D ;
END
```

FIG. 17

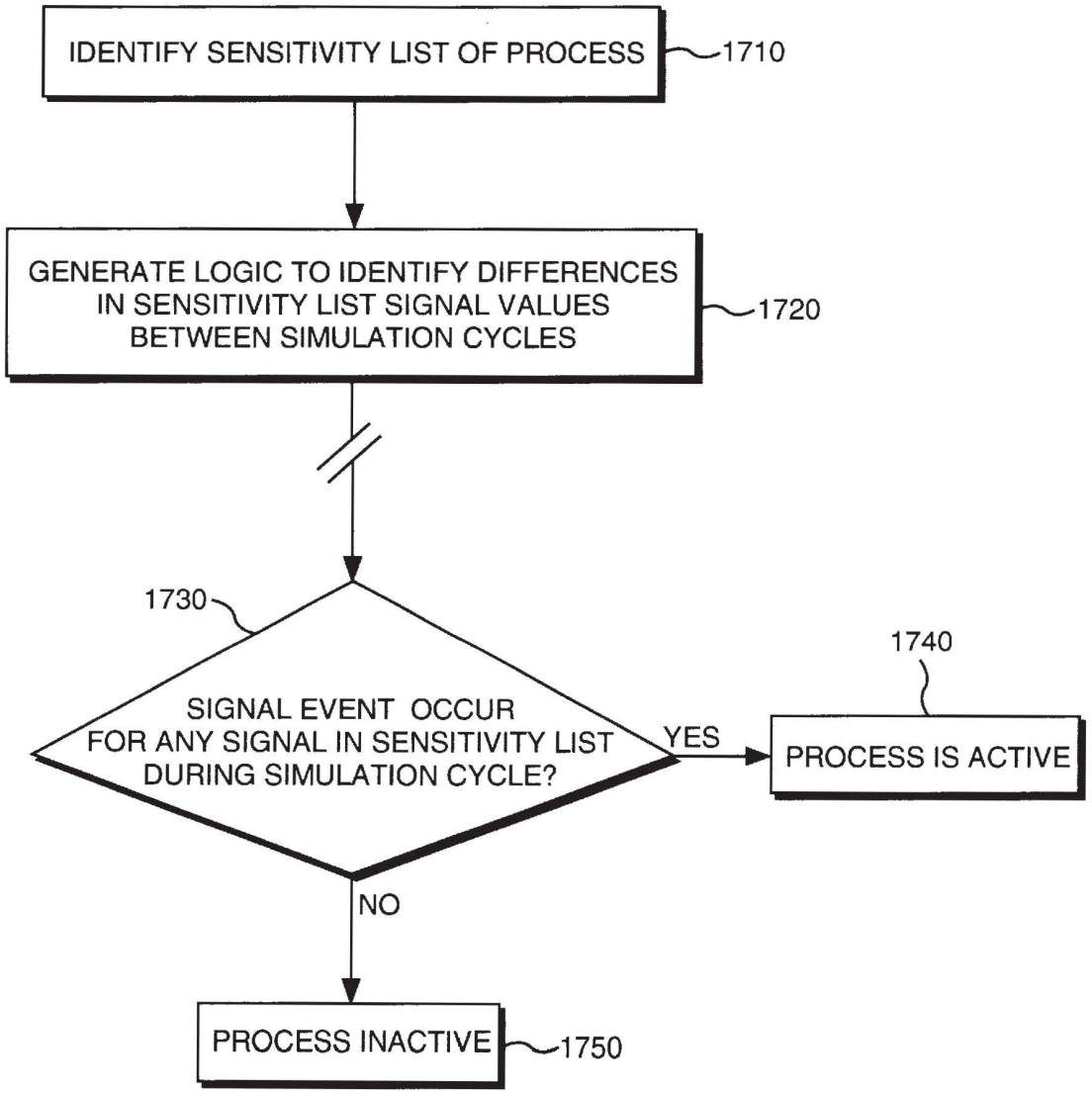

IDENTIFY SENSITIVITY LIST OF PROCESS
1710
GENERATE LOGIC TO IDENTIFY DIFFERENCES IN SENSITIVITY LIST SIGNAL VALUES BETWEEN SIMULATION CYCLES
1720
1730
SIGNAL EVENT OCCUR FOR ANY SIGNAL IN SENSITIVITY LIST DURING SIMULATION CYCLE?
YES
1740
PROCESS IS ACTIVE
NO
PROCESS INACTIVE
1750

FIG. 18

P1: PROCESS (A,B,C)

```
PROCESS (FAST_CLK)
 BEGIN
          IF (FAST_ CLK'EVENT AND FAST_ CLK='1')
           THEN
                  SAMPLED_A <=A ;
                  SAMPLED_B <=B ;
                  SAMPLED_C <=C ;
          END IF
          END PROCESS;
```

1810

```
P1_ ACTIVE <= (SAMPLED_A /= A)
       OR (SAMPLED_B /= B)
       OR (SAMPLED_C /= C);
```

1820

FIG. 19

```
CASE OPCODE IS
    WHEN "00" => TRACE1 := 1;
                 STATE := 1;
    WHEN "01" => TRACE2 := 1;
                 STATE:= 2 ;
    WHEN "10" => TRACE3 := 1;
                 STATE := 2 ;
    WHEN "11" => TRACE4 := 1;
                 STATE := 1 ;
END CASE ;
```

1910

FIG. 20

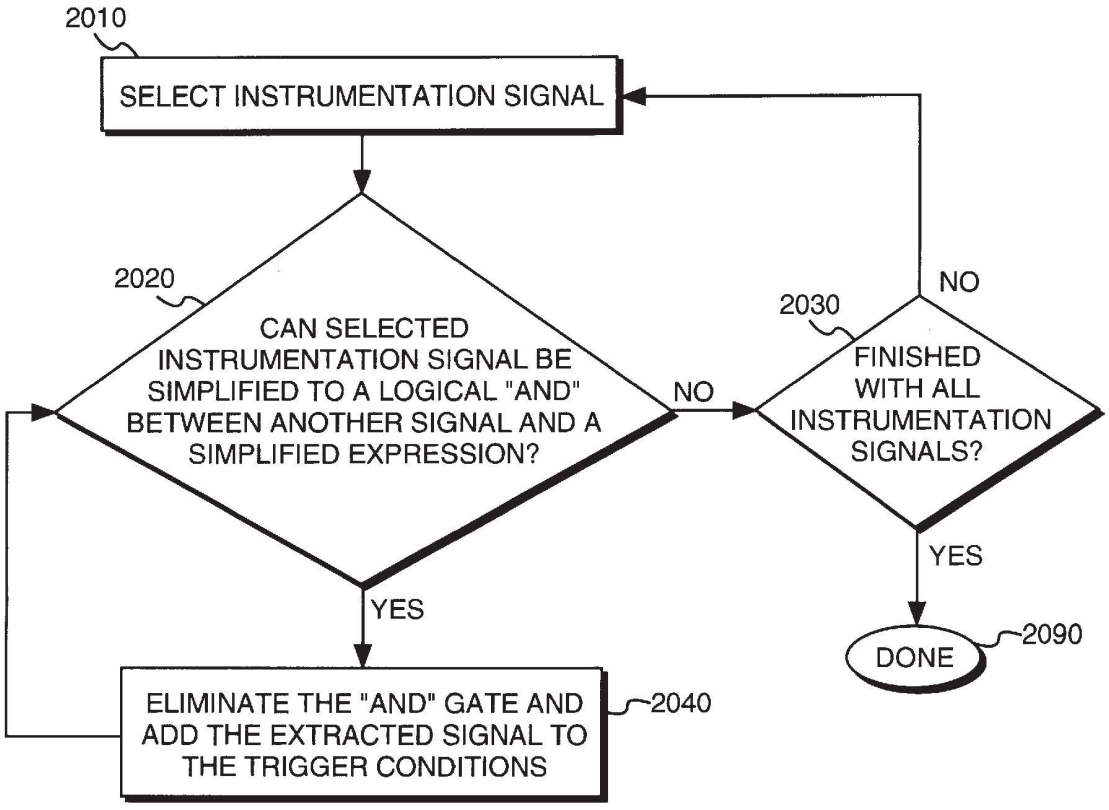


FIG. 21

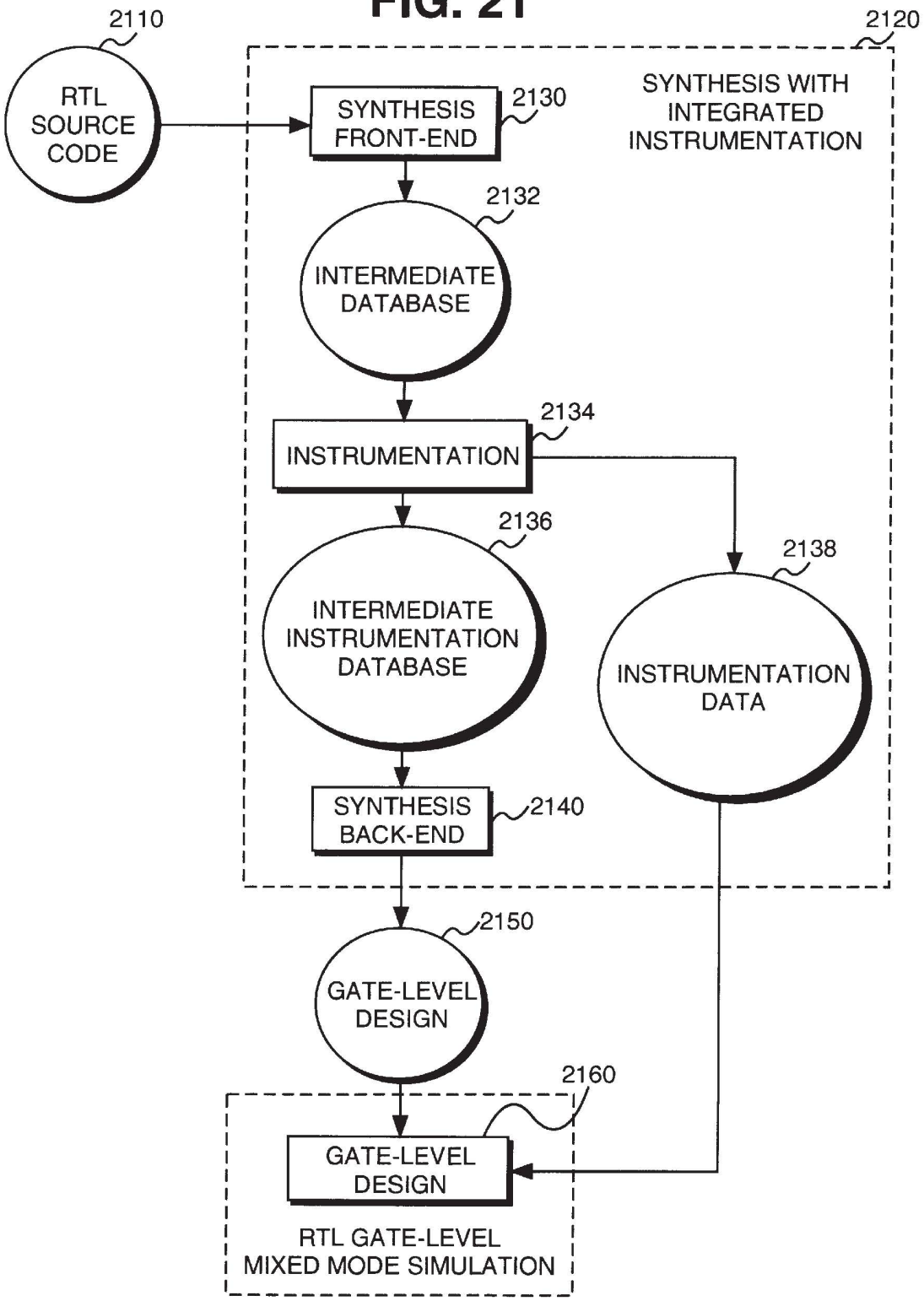
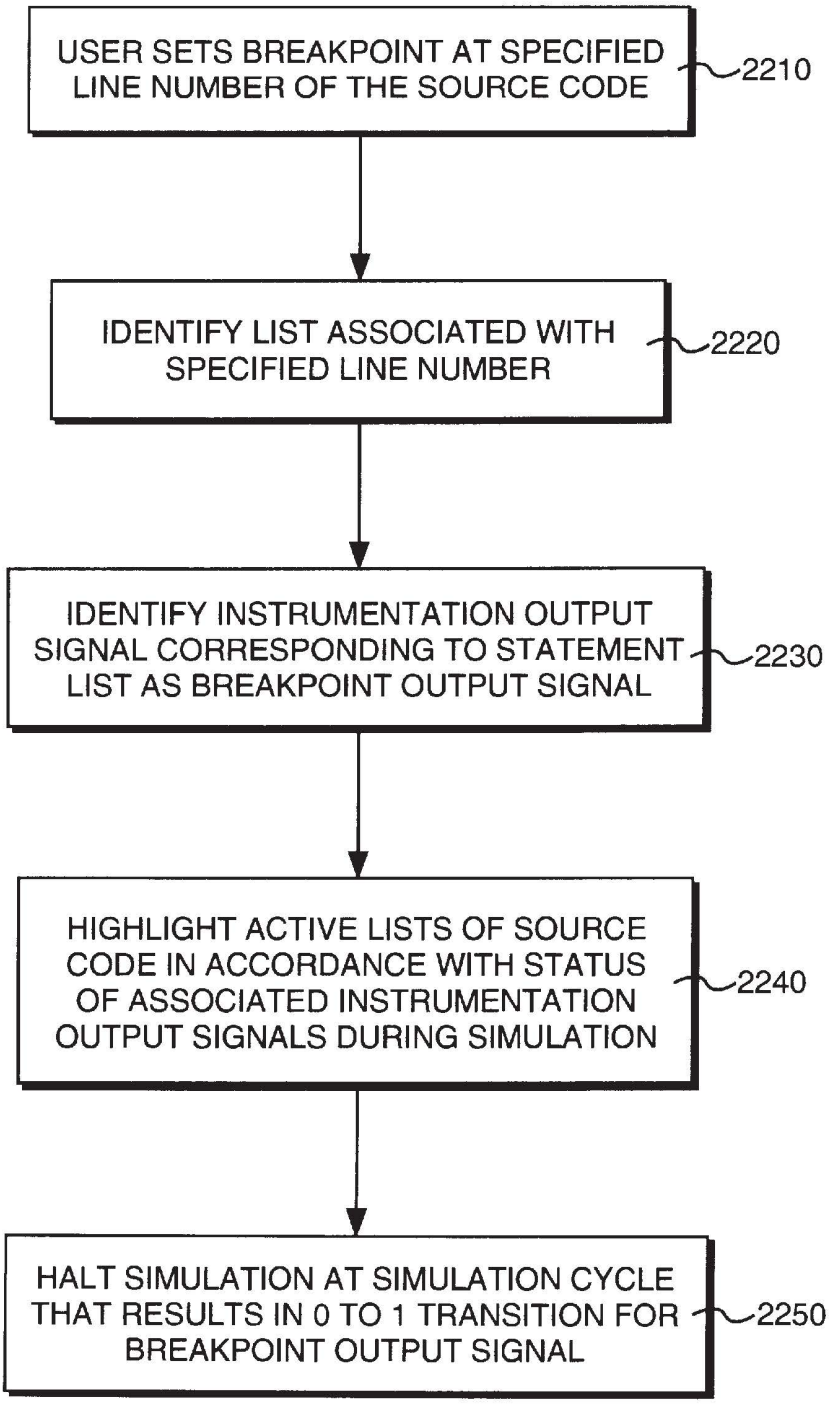

2110
RTL SOURCE CODE
2120
SYNTHESIS WITH INTEGRATED INSTRUMENTATION
SYNTHESIS FRONT-END
2130
2132
INTERMEDIATE DATABASE
2134
INSTRUMENTATION
2136
INTERMEDIATE INSTRUMENTATION DATABASE
2138
INSTRUMENTATION DATA
SYNTHESIS BACK-END
2140
2150
GATE-LEVEL DESIGN
2160
GATE-LEVEL DESIGN
RTL GATE-LEVEL MIXED MODE SIMULATION

FIG. 22


USER SETS BREAKPOINT AT SPECIFIED LINE NUMBER OF THE SOURCE CODE
2210
IDENTIFY LIST ASSOCIATED WITH SPECIFIED LINE NUMBER
2220
IDENTIFY INSTRUMENTATION OUTPUT SIGNAL CORRESPONDING TO STATEMENT LIST AS BREAKPOINT OUTPUT SIGNAL
2230
HIGHLIGHT ACTIVE LISTS OF SOURCE CODE IN ACCORDANCE WITH STATUS OF ASSOCIATED INSTRUMENTATION OUTPUT SIGNALS DURING SIMULATION
2240
HALT SIMULATION AT SIMULATION CYCLE THAT RESULTS IN 0 TO 1 TRANSITION FOR BREAKPOINT OUTPUT SIGNAL
2250

METHOD AND APPARATUS FOR GATE-LEVEL SIMULATION OF SYNTHESIZED REGISTER TRANSFER LEVEL DESIGNS WITH SOURCE-LEVEL DEBUGGING

This is a continuation in part of application Ser. No. 09/122,493, filed Jul. 4, 1998.

FIELD OF THE INVENTION

This invention relates to the fields of simulation and prototyping hen designing integrated circuits. In particular, this invention is drawn to debugging synthesizable code at the register transfer level during gate-level simulation.

BACKGROUND OF THE INVENTION

Integrated circuit designers have adopted the use of high-level hardware description languages due in part to the size and complexity of modem integrated circuits. One such description language is Very High Speed Integrated Circuit (VHSIC) Description Language, or VHDL. Further information regarding VHDL may be found in the IEEE Standard VHDL Language Reference Manual (IEEE 1076–1987, IEEE 1076–1993). Another such description language is Verilog. These high level description languages are typically generically referred to as hardware description languages (HDLs).

Synthesis is the process of generating a gate-level netlist from the high level description languages. Presently, synthesis tools recognize a subset of the high-level description language source code referred to as Register Transfer Level (RTL) source code. Further information regarding RTL source code may be found in the IEEE 1076.6/D1.10 Draft Standard for VHDL Register Transfer Level Synthesis (1997).

The RTL source code can be synthesized into a gate-level netlist. The gate-level netlist can be verified using gate-level simulation. The gate-level simulation can be performed using a software gate-level simulator. alternatively, the gate-level simulation may be performed by converting the ate-level netlist into a format suitable for programming an emulator, a hardware accelerator, or a rapid-prototyping system so that the digital circuit description can take an actual operating hardware form.

Debugging environments for high-level hardware description languages frequently include a number of functionalities for analyzing and verifying the design when performing simulation. For example, a designer can typically navigate the design hierarchy, view the RTh source code, and set breakpoints on a statement of RTL source code to stop the simulation. Statements are usually identified by their line number in the RTL source code. In addition, the debugging environment often supports viewing and tracing variables and signal values. The RTL simulation environment typically offers such RTL debugging functionalities.

RTL simulation is typically performed by using software RTL simulators which provide good flexibility. However, for complex designs, a very large number of test vectors may need to be applied in order to adequately verify the design. This can take a considerable amount of time using software RTL simulation as contrasted with hardware acceleration or emulation starting from a gate-level netlist representation (i.e., "gate-level hardware acceleration," or "gate level emulation"). Furthermore, it may be useful to perform in-situ verification, which consists of validating the design under test by connecting the emulator or hardware accelerator to the target system environment (where the design is to be inserted after the design is completed).

One disadvantage with gate-level simulation, however, is that most of the high-level information from the RTL source code is lost. Without the high-level information, many of the debugging functionalities are unavailable.

For example, the designer typically cannot set a breakpoint from the source code during gate-level simulation. Although signals can be analyzed during gate-level simulation, mapping signal values to particular source code lines can be difficult, if not impossible. If the source code is translated into a combinatorial logic netlist, for example, the designer cannot "step" through the source code to trace variable values. Instead, the designer is limited to analyzing the input vector and resulting output vector values. Although the signals at the inputs and outputs of the various gates may be traced or modified, these values are determined concurrently in a combinatorial network and thus such analysis is not readily mappable to the RTL source code.

A typical design flow will include creating a design at the RTL level, then synthesizing it into a gate-level netlist. Although simulation of this netlist can be performed at greater speeds using emulators or hardware accelerators, the ability to debug the design at the gate level is severely limited in comparison with software RTL simulation.

SUMMARY OF THE INVENTION

Methods of instrumenting synthesizable register transfer level (RTL) source code to enable debugging support akin to high-level language programming environments for gate-level simulation are provided.

One method of facilitating gate-level simulation includes the step of generating cross-reference instrumentation data including instrumentation logic indicative of the execution status of at least one synthesizable statement within the RTL source code. A gate-level netlist is synthesized from the RTL source code. Evaluation of the instrumentation logic during simulation of the gate-level netlist enables RTL debugging by indicating the execution status of the cross-referenced synthesizable statement in the RTh source code.

In one embodiment, the gate-level netlist is modified to provide instrumentation signals implementing the instrumentation logic and corresponding to synthesizable statements within the RTL source code. In various embodiments, this may be accomplished by modifying the RTL source code or by generating the modified gate-level netlist during synthesis as if the source code had been modified.

Alternatively, the gate-level netlist is not modified but the instrumentation signals implementing the instrumentation logic are contained in a cross-reference instrumentation database. In either case, the instrumentation signals indicate the execution status of the corresponding cross-referenced synthesizable statement. The instrumentation signals can be used to facilitate source code analysis, breakpoint debugging, and visual tracing of the source code execution path during gate-level simulation.

For example, a breakpoint can be set at a selected statement of the source code. A simulation breakpoint is set so that the simulation is halted at a simulation cycle where the value of the instrumentation signals indicate that the statement has become active .

With respect to visually tracing the source code during execution, the instrumentation logic is evaluated during gate-level simulation to determine a list of at least one active statement. The active statement is displayed as a highlighted statement.

With respect to source code analysis, cross-reference instrumentation data including the instrumentation signals

can be used to count the number of times a corresponding statement is executed in the source code. For example, an execution count of the cross-referenced synthesizable statement is incremented when evaluation of the corresponding instrumentation logic indicates that the cross-referenced synthesizable statement is active.

Other features and advantages of the present invention will be apparent from the accompanying drawings and from the detailed description that follows below.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings, in which like references indicate similar elements and in which:

FIG. **1** illustrates the process of synthesizing RTL source code into a gate-level design.

FIG. **2** illustrates one embodiment of a modified process for generating a gate-level design.

FIG. **3** illustrates one embodiment of a method for instrumenting level-sensitive RTL source code.

FIG. **4** illustrates VHDL source code.

FIG. **5** illustrates the gate-level design synthesized from the RTL source code of FIG. **4**.

FIG. **6** illustrates the VHDL source code of FIG. **4** modified in accordance with the method of FIG. **3**.

FIG. **7** illustrates one embodiment of the gate-level logic synthesized from the modified RTL source code.

FIG. **8** illustrates sample Verilog source code before instrumentation.

FIG. **9** illustrates the Verilog source of FIG. **8** instrumented in accordance with the method of FIG. **3**.

FIG. **10** illustrates the gate-level logic synthesized from the instrumented Verilog source code of FIG. **9**.

FIG. **11** illustrates VHDL source code for a D flip-flop with asynchronous reset.

FIG. **12** illustrates one method of instrumenting event-sensitive RTL source code.

FIG. **13** illustrates the source code of FIG. **11** modified in accordance with the instrumentation process of FIG. **12**.

FIG. **14** illustrates the gate-level logic synthesized for the instrumented source code of FIG. **13**.

FIG. **15** illustrates Verilog source code for a D flip-flop with asynchronous reset.

FIG. **16** illustrates the Verilog source code of FIG. **15** after instrumentation in accordance with the method of FIG. **12**.

FIG. **17** illustrates a method of instrumenting process activation.

FIG. **18** illustrates source code modified in accordance with the method of FIG. **17**.

FIG. **19** illustrates an instrumented "case" statement.

FIG. **20** illustrates a process for decreasing the logic needed to instrument the source code.

FIG. **21** illustrates incorporating instrumentation within the synthesis process.

FIG. **22** illustrates a method of setting a breakpoint in RTL source code for use during gate-level simulation.

DETAILED DESCRIPTION

FIG. **1** illustrates a typical RTL source code synthesis process. HDL code including synthesizable RTL source code (**110**) serves as input to a synthesis process **120**. In one embodiment, the RTL source code **110** is synthesized in step **140** to produce a gate-level design **150**. The gatelevel design can be used for gate-level simulation as illustrated in step **160**.

Typically the gate-level design comprises a hierarchical or flattened gate level netlist representing the circuit to be simulated. The various signals in a design are referred to as nets. A hierarchical netlist is made of a list of blocks, whereas a flattened netlist comprises only one block. A block contains components and a description of their interconnection using nets. Components can be reduced to combinatorial or sequential logic gates, or they may be hierarchical blocks of lower level.

For example, the component may be a primitive gate denoting a single combinatorial logic function (e.g., AND, NAND, NOR, OR, XOR, NXOR, etc.) or a single storage element such as a flip-flop or latch for sequential logic. One example of a set of primitive gates is found in the generic library GTECH available from Synopsys, Inc. of Mountain View, Calif.

Alternatively the component may be an application specific integrated circuit (ASIC) library cell which can be represented by a set of primitive gates. One example of an ASIC library is the LCA300K ASIC library developed by LSI Logic, Inc., Milpitas, Calif.

A component may also be a programmable primitive that represents a set of logic functions and storage. One example of a programmable primitive is the configurable logic block (CLB) as described in The Programmable Gate Array Handbook, Xilinx Inc., San Jose, 1993.

Another example of a component is a macro block denoting a complex logic function such as memories, counters, shifters, adders, multipliers, etc. Each of these can be further reduced to primitive gates forming combinatorial or sequential logic.

Three major categories of tools are available to the designer to simulate and test the design. Software RTL simulators (such as ModelSim™ from Model Technology, Inc.) typically offer a high-level of abstraction for their debugging environment, but have limited performance in terms of speed and no in-situ capacity. Software gate-level simulators (such as QuickSim™ from Mentor Graphics Corporation) typically offer limited level of abstraction and speed as well as no in-situ capacity. Hardware gatelevel simulators (such as Cobalt™ and System Realizer™ from Quickturn Inc., Avatar™ from Ikos, and fast-prototyping systems usually built from FPGAs) typically offer very good performance in terms of speed and in-situ capacity, but a limited debugging environment.

When testing the design described by the HDL source code a designer may choose to simulate and validate the design at the RTL source code level (i.e., RTL simulation). RTL simulation typically permits the designer to set breakpoints in the source code, navigate the design hierarchy, view variables and signals and trace the value of these variables and signals.

When testing complex designs, millions or billions of test vectors may need to be applied in order to adequately test the design. Hardware accelerators or emulators can be used with the gate-level design to test the design at a much greater speed than what is typically possible through software simulation (i.e. either software RTL simulation or software gate-level simulation). Unfortunately, the gate-level design generated in step 150 typically includes none of the high-level information available in the RTL source code **110**. As a result, features available during RTL simulation such as

setting breakpoints or analyzing the source code coverage are not available during gate-level simulation.

Instrumentation is the process of preserving high-level information through the synthesis process. Instrumentation permits simulation of a gatelevel netlist at the level of abstraction of RTL simulation by preserving some of the information available at the source code level through the synthesis process.

FIG. **2** illustrates one embodiment of the instrumentation process in which instrumentation is integrated with the synthesis process. RTL source code **210** is provided to the synthesis process **220**. The synthesis process **120** of FIG. **1** has been modified to include an instrumentation step **234**. After instrumentation the instrumented code is then synthesized in step **240** as the original RTL source code was in step **140** of FIG. **1**.

In one embodiment, instrumentation results in generating a modified gate-level design to permit reconstitution of the flow of execution of the original RTL source code during gate-level simulation. Generally instrumentation logic is created for a synthesizable statement in the RTL source code either by modifying the RTL source code or by analyzing the RTL source code during the synthesis process. The instrumentation logic provides an output signal indicative of whether the corresponding synthesizable statement is active. A gate-level design including the instrumentation output signal is then synthesized. Referring to FIG. **2**, the resulting gate-level design **250** contains additional logic to create the additional instrumentation output signals referenced in instrumentation data **238**.

In an alternative embodiment, the RTL source code is analyzed to generate a cross-reference database as instrumentation data **238** without modifying the gate-level design. The cross-reference database indicates the combination of already existing signals in the form of instrumentation logic that can be evaluated during simulation to determine whether a particular line of the RTL source code is active. The cross-reference database contains a cross-reference between these instrumentation logic output signals and the position of the corresponding statement in the source code. The instrumentation data **238** is likely to contain considerably more complex logic to evaluate during simulation when the approach of not modifying the gate-level design (i.e., "pure" cross-reference database) is taken.

The two approaches have tradeoffs. The gate-level design modification technique does not require special knowledge of the target simulation environment. Moreover, the gate-level design modification technique significantly reduces or eliminates the complexity of the logic to be evaluated during simulation to the extent that emulator or accelerator hardware triggering circuitry can be used to take an action when the corresponding statement is executed.

For example, the hardware triggering circuitry may be used to halt the simulation at a particular statement or to count the number of times a particular statement is executed. The resulting gate-level design used during simulation, however, will not be the design actually used for production thus simulation may not verify accurately the behavior of the gate-level design used for production. Furthermore, simulation of modified gate-level design may require more physical resources in hardware than the original design alone if gates have been added in order to implement the instrumentation logic.

Alternatively, the pure cross-reference database technique typically results in greater complexity of instrumentation logic to evaluate during simulation, but does not otherwise affect the original gate-level design. The greater complexity, however, may prevent the use of the hardware triggering circuitry to halt the simulation or to track source code coverage. Thus the pure cross-reference database technique may result in a significantly slower simulation time. Furthermore, since the evaluation may be performed by software, direct verification of the gate-level design in the target system through in-situ verification may not be possible. The instrumentation data including the logic added for instrumentation purposes can be eliminated after testing, however, without disrupting the gate-level design.

In essence the gate-level design modification technique greatly simplifies the analysis and the instrumentation logic required for cross-referencing by modifying the gate-level design to create unique signals and therefore simpler logic to evaluate (i.e., a single signal). The resulting instrumentation logic cross-referenced in the instrumentation data **238** is easily evaluated during simulation. Various embodiments of instrumentation may combine the gate-level design modification technique or the pure cross-referencing technique in order to trade off simulation speed, density, and verification accuracy.

If the gate-level simulator, hardware accelerator, or emulator (e.g., through the use of a logic analyzer which can be external to the emulator) has the capacity to set breakpoints whenever certain signals reach a given value, then it is possible to implement breakpoints corresponding to RTL simulation breakpoints in the gate-level design. Whenever the user specifies a breakpoint in the RTL source code, the condition can be converted to a comparison with key signals in the gate-level design.

Instrumentation data **238** identifies the RTL source code statements each instrumentation output signal is associated with. Instrumentation data **238** is generated during the instrumentation process of step **234**. In one embodiment, the instrumentation data is implemented as gates that can then be simulated by the target-level simulator. By examining the state of each instrumentation output signal during gate-level simulation, the user can determine which portions of RTL source code are being simulated. This in turn permits the designer to determine RTL source code coverage. By tracking the instrumentation signal values for each cycle of execution, the designer can determine how many times each line of the RTL source code has been activated.

The instrumentation data **238** can be used during simulation to ensure every possible state transition has been tested. For example, a Finite State Machine analyzer can determine from the values of the instrumentation output signals whether every possible state transition has been tested.

The instrumentation data **238** can also be used to enhance the source code display. In one embodiment, the source code is repositioned on the display so as to indicate the execution paths that are active during a current cycle. In another embodiment, the active source code in a given cycle is highlighted to indicate that it is active. This permits the designer to visually see the process flow without having to determine the value of each signal. In one embodiment, the instrumentation data **238** is used to enhance the display of the original RTL source code rather than the source code resulting from instrumentation.

An integrated circuit design is typically built by assembling hierarchical blocks. In VHDL, a block corresponds to an entity and architecture. In Verilog, a block corresponds to a module. In both HDLs, a block typically includes a declarative portion and a statement portion. The declarative portion generally includes the list of the ports or connectors.

The statement portion describes the block's behavior and is typically where a designer needs help when debugging a design. The statement portion includes concurrent signal assignment statements and sequential statements.

Concurrent signal assignment statements assign a logic expression to a signal. The signal is typically available for viewing at all times and thus breakpoints can be set in accordance with when the signals reach a certain value.

Sequential statements assign values depending upon the execution flow of the sequence. Sequential statement analysis is typically where the designer needs the greatest aids in debugging the design.

Sequential statements are typically found in VHDL "processes" and in Verilog "always" blocks. Processes or always blocks can be built of an unlimited combination of sequential statements including loops, conditional statements, and alternatives. There are at least two classes of sequential statements: level-sensitive and event-sensitive. Level-sensitive sequential statements only depend on the value of the inputs and can be synthesized to logic networks of combinatorial gates and latches. Event-sensitive sequential statements additionally require sequential logic such as flip-flops.

In one embodiment, level-sensitive RTL source code is instrumented by creating and associating one output signal with each list of synthesizable sequential statements. A list can consist of one or more sequential statements.

In one embodiment, each statement is a list. In an alternative embodiment, each list corresponds to a branch of the RTL source code. A list corresponding to a branch typically comprises a plurality of adjacent sequential statements, but may comprise a single sequential statement. Only one output signal is needed for each list of synthesizable sequential statements in a branch rather than for every sequential statement in the source code. Examples of sequential statements that create branches in the RTL source code are conditional statements such as IF-THEN statements and SELECT-CASE statements.

FIG. **3** illustrates one method of modifying RTL source code for level-sensitive code. Generally, a unique local variable is created for each list of adjacent sequential statements in step **310**. The level sensitive code instrumentation includes the step of modifying the RTL source code to initialize each of these unique variables to zero at the beginning of the process being instrumented in step **320**. One unique variable assignment statement is inserted into each list of adjacent sequential statements corresponding to an executable branch in step **330**. The assignment statement sets the unique variable to one. At the end of the process all the unique local variables are assigned to global signals in step **340**. Steps **310** and **320** are more generically referred to as initialization. Step **330** is referred to as flow instrumentation. Step **340** is referred to as "gathering."

FIG. **4** illustrates non-instrumented VHDL source code. The VHDL source code **400** includes nine sequential statements within the process block. Eight of these nine statements are non-signal assignment sequential statements. These eight sequential statements form six statement lists or executable branches of the code. IF-THEN statement **410** comprises one list. Signal assignment statement **420** comprises a second list. Statements **430**, **440**, **450** and **490** comprise a third list because they would be executed sequentially within the same execution path. Statements **460**, **470**, and **480** form individual lists.

FIG. **5** illustrates one embodiment of the logic **500** resulting from the synthesis of the RTL source code of FIG. **4**. This figure may be used for comparison with the gate level design generated from instrumented code described below.

FIG. **6** illustrates the source code of FIG. **4** after instrumentation as described in FIG. **3**. The added statements are italicized for emphasis. For example, line **612** has been added to the source code to create six unique local variables (TRACE**1** through TRACE**6**), one for each of the six identified lists, in accordance with step **310** of FIG. **3**.

In accordance with step **330** of FIG. **3**, a trace variable assignment statement has been added adjacent to each of the lists. Referring to FIGS. **4** and **6**, variable assignment statement **630** has been added adjacent to the first list comprising statement **410**. Variable assignment statement **632** has been added adjacent to the second list comprising statement **420**. Variable assignment statement **634** has been added adjacent to the third list comprising statements **430**, **440**, **450** and **490**. Variable assignment statement **636** has been added adjacent to the fourth list comprising statement **460**. Similarly, variable assignment statements **638** and **640** have been added adjacent to the fifth list comprising statement **470** and the sixth list comprising **480**, respectively. Each of variable assignment statements **630** through **640** assigns a unique local variable the value of one.

Code portion **620** is added to initialize the unique local variables to zero at the beginning of the process in accordance with step **320** of FIG. **3**.

Each of the local variables is assigned to a global output signal in accordance with step **340** of FIG. **3** by code portion **650**. If required by the HDL, the global signals are declared by code portion **610**. Similarly, the trace variables are declared by code portion **612**.

In one embodiment, the unique local variables can actually be a single array where each "unique variable" or trace variable corresponds to a different position in the array. Similarly, in one embodiment, the additional global signals are described by an array where each of the global signals is represented by a different index of the array.

Coding practices for VHDL generally require variables to be used within the process and a signal assignment at the end of the process to propagate the variable values at the end of the process. In one embodiment, markers such as variable assignment statements are used to track the execution paths. Markers such as variable assignment statements are not typically synthesized into logic indicating the variable values, thus the variable assignment statements are used in conjunction with signal assignment statements in order to produce signals indicating whether various portions of the synthesized code are being executed.

If permitted by the HDL, however, global signal assignments can be used in lieu of local variable assignment statements. This would simplify the process of FIG. **3** in that there would be no need to create or initialize local variables. In addition the step of assigning the local variables to global signals could be eliminated because values are assigned directly. The key is ensuring that there is a unique output signal created and associated with each list of sequential statements regardless of the coding practice used to achieve this goal.

FIG. **7** illustrates one embodiment of the logic **700** generated through instrumentation. In particular, FIG. **7** illustrates the additional gate-level logic added to generate signals SIG_TRACE**1** through SIG_TRACE**6** from synthesis of the modified source code.

FIG. **8** illustrates a Verilog "always" block **800**. FIG. **9** illustrates the same code after instrumentation in accordance with the process of FIG. **3**. Due to Verilog syntax

requirements, "BEGIN-END" statements were used to properly group the instrumentation variable with the other statements in each executable path.

Although the code of FIG. **8** results in a latch, application of the technique of FIG. **3** to the source code of FIG. **8** ensures that the instrumentation output signals are the result of combinatorial logic only. Thus the logic for determining which lines of code are active can be purely combinatorial even when the RTL source code is synthesized into latches.

FIG. **10** illustrates one embodiment of gate-level logic **1100** generated by synthesis of the instrumented "always" block **900** of FIG. **9**. The instrumentation signals SIG_TRACE**1**, SIG_TRACE**2**, SIG_TRACE**3**, and SIG_TRACE**4** are the result of combinatorial logic only.

Referring to FIG. **2**, the instrumentation data **238** can be stored in a cross-reference file. In one embodiment, the cross-reference file contains a mapping between original source code line numbers and instrumentation signals. Each time an instrumentation variable (and its associated signal) is added to the source code, all the line numbers of the statements in the list associated with the instrumentation variable are added to the file. This cross-reference file (i.e., instrumentation data **238**) can be used by the gate-level simulation environment to convert the designer's breakpoints into actual conditions on instrumentation signals.

A more sophisticated method than that illustrated in FIG. **3** is required to instrument RTL source code having references to signal events. Typically such source code is used to describe edge-sensitive devices. References to signal events typically imply flip-flops. A signal event is a signal transition. Thus any signal computed from a signal transition references a signal event.

FIG. **11** illustrates sample VHDL code **1100** with references to a signal event. VHDL code **1100** implements a D-type flip-flop with asynchronous reset. The event in this example is a transition on the clock signal (CLK) as referenced by the term "CLK'EVENT."

In accordance with VHDL specifications signals can have various attributes associated with them. A function attribute executes a named function on the associated signal to return a value. For example, when the simulator executes a statement such as CLK'EVENT, a function call is performed to check this property of the signal CLK. In particular, CLK'EVENT returns a Boolean value signifying a change in value on the signal CLK. Other classes of attributes include value attributes and range attributes.

In VHDL code **1100**, the signal CLK has a function attribute named "event" associated with it. The predicate CLK'EVENT is true if an event (i.e., signal transition) has occurred on the CLK signal. Assigning a value to a signal (i.e., a signal transaction) qualifies as an event only if the transaction results in a change in value or state for the signal. Thus the predicate CLK'EVENT is true whenever an event has occurred on the signal CLK in the most recent simulation cycle. The predicate "IF (CLK'EVENT and CLK='1')" is true on the rising edge of the signal CLK.

Depending upon the specifics of the HDL, another function such as RISING_EDGE(CLK) might be used to accomplish the same result without the use of attributes. The function RISING_EDGE(CLK) is still an event even though the term "event" does not appear in the function.

FIG. **12** illustrates a method of instrumenting source code having references to signal events. In step **1210**, every signal event is sampled using a fast clock. In other words, every signal whose state transition serves as the basis for the determination of another signal is sampled. An instrumentation signal event corresponding to the original signal event is generated in step **1220**. Any attributes of the original signal must similarly be reproduced based on the instrumentation signal if the source code uses attributes of the original signal event.

In step **1230**, every process that references a signal event is duplicated. In step **1240**, each list of sequential statements within the duplicate version of the code is replaced by a unique local variable assignment statement. In step **1250**, each time a signal event is referenced in the duplicated version of the code, it is replaced by the sampled signal event computed in step **1210**. The modified RTL source code can then be synthesized in step **1260** to generate gate-level logic including the instrumentation output signals.

FIG. **13** illustrates application of the method of FIG. **12** to the source code of FIG. **11**. In order to detect signal events properly for instrumentation, the signal events are sampled using a fast clock provided during gate-level simulation (i.e., FAST_CLK). FAST_CLK has a higher frequency than the CLK signal and thus permits detecting transition edges before signals depending upon CLK (including CLK itself) can.

The only signal event referenced in FIG. **11** is a transition in the signal CLK indicated by the term CLK'EVENT. Thus an instrumentation version of CLK'EVENT is created by sampling the signal CLK using FAST_CLK. The signal FAST_CLK has a higher frequency than the signal CLK.

Code portion **1310** samples the CLK signal on every rising edge of the signal FAST_CLK to generate a sampled version of the signal CLK named SAMPLED_CLK. The instrumentation version of CLK'EVENT is CLK_EVENT which is generated in code portion **1310** based on SAMPLED_CLK. The instrumentation signal CLX_EVENT (corresponding to CLK'EVENT) is determined by comparison of signals SAMPLED_CLK and CLK. The signal CLK_EVENT is true only when the signal SAMPLED_CLK is not the same as CLK, thus indicating a transition has occurred in the signal CLK.

Although not required for this example, code portion **1310** also illustrates the generation of instrumentation clock signal attributes based on SAMPLED_CLK. For example, the signal CLK'STABLE is the complement of CLK'EVENT. Thus code portion **1310** indicates the instrumentation version of the attribute CLK'STABLE (i.e., CLK_STABLE) computed on the instrumentation clock signal (i.e., SAMPLED_CLK). The signal CLK'LASTVALUE is a function signal attribute that returns the previous value of the signal CLK. The instrumentation version (i.e., CLK_LASTVALUE) of the attribute CLK'LASTVALUE is similarly computed on the instrumentation clock signal SAMPLED_CLK.

Although CLK_LASTVALUE is the same as the sampled clock signal, SAMPLED_CLK, code **1310** introduces the intermediate signal SAMPLED_CLK for purposes of illustrating sampling of the CLK signal. The signal CLK_LASTVALUE can be defined in lieu of SAMPLED_CLK in order to eliminate the introduction of an unnecessary intermediate signal SAMPLED_CLK and the subsequent step of assigning CLK_LASTVALUE the value of SAMPLED_CLK.

Neither CLK_LASTVALUE nor CLK_STABLE are needed in this example for code portion **1320**, however, code portion **1310** serves as an example of how to generate instrumentation versions of signal attributes typically used to describe edge-sensitive devices.

Code portion **1320** represents the instrumented duplicate of original code portion **1330**. The process of code portion

1330 references the event CLK'EVENT in the IF-ELSIF statement. In code portion 1320, all sequential statements (except the statement referencing an event) have been replaced with unique local variable assignment statements. These statements assign a local variable (i.e., TRACE1, TRACE2) the value "1." Code portion 1320 also includes statements to create and initialize these unique local variables.

In accordance with step 1240, every occurrence of a signal event is replaced with the sampled version of that event. Thus, for example, references to CLK'EVENT in code portion 1330 are replaced with references to CLK_EVENT in code portion 1320. Moreover, the process parameter list is modified to include the generated signal CLK_EVENT. FIG. 14 illustrates the gate-level logic 1400 resulting from synthesis of the code in FIG. 13.

FIG. 15 illustrates Verilog source code 1500 for a D flip-flop with asynchronous reset. FIG. 16 illustrates the code 1600 resulting from modifying source code 1500 in accordance with the method of FIG. 12.

One advantage of the instrumentation approach of FIG. 12 is that the gates generated by the synthesis tool are the same ones that would be generated if the source code had not been instrumented. The gates generated for the instrumentation logic are not intermingled with the gates generated from the non-instrumented source code. This permits design verification with gate-level logic that does not need to be re-verified after instrumentation verification. Thus the designer can verify the result of synthesis at the gate level while retaining RTL breakpoint feature. In some cases, however, the synthesis tool may not recognize that the same code appears twice. This may incur an additional relatively expensive phase of resource sharing in order to achieve the same performance results as the process illustrated in FIG. 3.

One advantage of the instrumentation process of FIG. 3 over that of FIG. 12, however, is that a synthesis tool can typically analyze the source code to detect obvious resource sharing.

The instrumentation methods of FIGS. 3 and 12 permit detecting any path that has been taken while a VHDL process or a Verilog "always" block is active. Tracking the activation of each process permits further analysis.

FIG. 17 illustrates a method of instrumenting the activation of the processes (or "always" blocks) themselves for subsequent determination of whether the process is active during gate-level simulation .

In step 1710, the sensitivity list of a process is identified. In step 1720, logic is generated to compare the signals in the sensitivity list between consecutive simulation cycles. Subsequently, during gate-level simulation in step 1730, a determination is made as to whether an event has occurred on any of the sensitivity list signals. Each simulation cycle that a signal indicates a difference (i.e., a signal event has occurred), the process is active as indicated by step 1740. Otherwise, if no events have occurred on any of the sensitivity list signals, the process is inactive as indicated by step 1750.

FIG. 18 illustrates the code added to determine if process P1 is active. The added code is italicized. The sensitivity list of process P1 includes signals a, b, and c. In accordance with step 1720 of FIG. 17, code section 1810 creates sampled versions of a, b, and c using FAST_CLK as described above. The sampled versions of a, b, and c are SAMPLED_A, SAMPLED_B, and SAMPLED_C, respectively.

Code section 1820 determines if an event has occurred on each of the sensitivity list signals. The test "(SAMPLED_A /=A)" is true if an event occurs with respect to signal A. Similarly "(SAMPLED_B /=B)" and "(SAMPLED_C /=C)" indicate whether an event has occurred with respect to signals B and C. Process P1 is active if any one of these tests is true. Thus the variable P1_ACTIVE is generated by combining each of these signal events using the logical OR function in code section 1820. Thus signal P1_ACTIVE indicates whether process P1 is active.

Process instrumentation data can be added to the instrumentation data cross-reference file in order to enhance the source code display. For example, the active process in a given cycle can be highlighted to indicate it is active. This permits the designer to visually see the active processes without having to determine the value of each signal. In one embodiment, the instrumentation data is used to enhance the display of the original RTL source code rather than the source code resulting from instrumentation.

The instrumentation techniques presented result in gate level designs providing explicit instrumentation signals to indicate that some specific portion of the source code is active. The number of instrumentation signals tends to increase with the complexity of the system being modeled.

Some optimizations may be performed to decrease the number of instrumentation signals. At least one execution path will be active any time a process is activated. As a result, the TRACE1 variable in the examples of FIGS. 6 and 9 tend to provide no additional information and thus SIG_TRACE1 is somewhat trivial as can be seen from the synthesized logic of FIGS. 7 and 10. Thus at least one trace variable (and therefore one output signal trace) can typically be eliminated.

In some cases the execution status of each branch of the code can be determined even though every branch is not explicitly instrumented. To verify the execution status of every branch, the instrumentation process need only ensure that each branch is instrumented either explicitly or implicitly through the instrumentation of other branches.

In some instances, the capacity of hardware triggers can be used to eliminate some of the instrumentation by combining several signals into one condition. The number of gates simulated can be reduced by replacing logical AND conditions that appear in the equations of instrumentation signals by simulator-specific triggers.

For example, consider the instrumented CASE statement code fragment 1910 illustrated in FIG. 19. For purposes of example, only the trace variable assignment statements are shown for the four possible cases. A synthesis tool will generate four comparisons with the vector "opcode." Each trace variable is associated with one of the possible values of opcode. Clearly, however, the additional logic is unnecessary because setting a breakpoint on any one of the case conditions corresponds to setting a trigger on the vector for the corresponding value of "opcode."

FIG. 20 illustrates a method for optimizing the instrumentation process. In particular, an instrumentation signal is selected in step 2010. In step 2020, a determination is made to whether the equation of the current signal can be expressed as a logical AND between a signal and a simplified expression. If so, then the AND gate should be eliminated in step 2030 and the extracted signal can be added to the trigger conditions during simulation in step 2040. If triggers can be activated on zeroes as well as ones, then step 2020 can also determine whether an equation can be simplified as a logical negation of a subexpression and the logical negation of the subexpression can be added to the trigger conditions during simulation in step 2040 where

appropriate. Step **2020** would then be applied recursively until the equation cannot be further simplified. This process is then applied to all of the instrumentation signals.

For example, signal TRACE**4** is the result of performing a logical AND between opcode(**0**) and opcode(**1**). Thus TRACE**4** is active only when opcode ="11". In accordance with FIG. **20**, the AND gate can be removed and the simulator trigger conditions would be changed from TRACE**4**=1 to "OPCODE(**0**)=1 AND OPCODE(**1**)=1." This process would then be applied recursively to all signals remaining in the trigger condition. Thus if OPCODE(**0**) happened to be the result of an AND between two other signals, the AND gate could again be eliminated from the synthesized gate-level design and the trigger conditions could be updated accordingly as long as no other signals used "OPCODE(**0**)" as an input. If no other logic uses "OPCODE(**0**)" as an input, then the trigger conditions can be updated to refer to the signals used to generate OPCODE (**0**) and the gate-level netlist AND gate can safely be eliminated. More generally, any optimization that consists of eliminating gates and other elements by transferring the implementation of the instrumentation logic to the logic analyzer of the target simulator can be performed.

Where permitted by the gate-level simulator, the instrumentation required for detecting activation of a process may similarly be reduced. In particular, greater efficiency may be possible by keeping a list of all the signals in the process sensitivity list and then testing whether events occurred on the signals in the sensitivity list. Further optimization may be made possible by sharing the logic for signals that appear on the sensitivity list of more than one process. The original signal can be sampled once initially. A comparison is made between the initial value and the current value of the signal to generate an event signal indicative of whether an event has occurred on that signal. The event signal can then be used for instrumentation of processes with events and for tracking process activation.

FIGS. **3**, **12**, and **17** illustrate methods of modifying the original RTL source code for instrumenting processes and level-sensitive and edge-sensitive source code. Trace variables (i.e., instrumentation variables) can be used to track the execution of any path within the source code. Additional output signals are generated from instrumentation variables in order to detect the execution paths of the source code. In the illustrated embodiments, the instrumentation variables are reset at the beginning of a process and the signals are assigned at the end of the process in order to ensure that all the signals are assigned regardless of which execution path is taken inside the process.

In an alternative embodiment, the signals might be directly assigned in the execution path of the process. Typically, this alternative embodiment would force the synthesis tool to generate complicated structures including latches due to the nature of HDLs and simulation rules.

The methods of FIGS. **3**,**12**, and **17** can be applied to the source code before the source code is synthesized. Thus in one embodiment the steps that modify the RTL source code can be performed before but entirely independently of the synthesis process itself.

FIG. **21** illustrates an embodiment in which the instrumentation data is generated entirely within the synthesis process. The process of creating output signals associated with synthesizable statements in the source code and then synthesizing the source code into a gate-level design including the output signal can be incorporated into the synthesis tool itself so that modification of the RTL source code is not required.

For example, one of the steps performed by a synthesis tool for generation of the gate-level design is parsing the RTL source code. Parsing the RTL source code results in a parser data structure that is subsequently used to generate the gate-level design. Instead of modifying the source code, the synthesis tool can simply set markers inside the parser data structure.

FIG. **22** illustrates one application of using the instrumentation signals for tracing execution flow using breakpoints. In step **2210**, the user sets a breakpoint at a specified line number of the source code. The specified line number is then associated with one of the instrumented lists of statements in step **2220**. In step **2230**, the instrumentation signal for the associated list is identified as the breakpoint output signal.

During the gate-level simulation run, the active lists (identified by transitions in their corresponding instrumentation signals) may be highlighted and displayed for the user as indicated in step **2240**. For example, the active lists may be portrayed in a different color than the inactive lists. Alternatively, the active lists may be displayed using blinking characters, for example. The instrumentation data file can be used to associate an instrumentation signal with a list of source code line numbers to be highlighted.

In response to a 0 to 1 transition in the breakpoint output signal, the simulation can be stopped as indicated in step **2250**. Thus through instrumentation the designer has the ability to effectively set breakpoints in the RTL source code which can be acted upon during RTL simulation.

The methods of instrumentation may be implemented by a processor responding to a series of instructions. In various embodiments, these instructions may be stored in a computer system's memory such as random access memory or read only memory.

The instructions may be distributed on a nonvolatile storage medium for subsequent access and execution by the processor. Typically the instructions are stored in the storage medium for distribution to a user. The instructions may exist in an application program form or as a file stored in the storage medium. The instructions are transferred from the nonvolatile storage medium to a computer system for execution by the processor.

In one embodiment, the program or file is installed from the storage medium to the computer system such that the copy of the instructions in the nonvolatile storage medium is not necessary for performing instrumentation. In another embodiment, the program or file is configured such that the original nonvolatile storage medium is required whenever the instructions are executed.

Nonvolatile storage mediums based on magnetic, optical, or semiconductor memory storage principles are readily available. Nonvolatile magnetic storage mediums include floppy disks and magnetic tape, for example. Nonvolatile optical storage mediums include compact discs, digital video disks, etc. Semiconductor-based nonvolatile memories include rewritable flash memory.

Instrumentation allows the designer to perform gate-level simulation of synthesized RTL designs with source-level debugging. In addition, the instrumentation process allows the designer to examine source code coverage during simulation.

In the preceding detailed description, the invention is described with reference to specific exemplary embodiments thereof. Various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. A method comprising the steps of:

a) identifying at least one statement within a register transfer level (RTL) synthesizable source code; and

b) synthesizing the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of the at least one statement.

2. The method of claim 1 wherein step b) includes the step of:

i) generating instrumentation logic to provide the instrumentation signal as if the source code included a corresponding signal assignment statement within a same executable branch of the source code as the identified statement.

3. The method of claim 1 wherein step b) includes the steps of:

i) initializing a marker to a first value at the beginning of a process within the source code; and

ii) setting the marker to a second value within a same executable branch of the source code as the identified statement.

4. The method of claim 3 further comprising the step of:

iii) assigning the value of the marker to the instrumentation signal at the end of the process.

5. A method of generating a gate level design, comprising the steps of:

a) creating an instrumentation signal associated with at least one synthesizable statement contained in a register transfer level (RTL) synthesizable source code; and

b) synthesizing the source code into a gate-level design having the instrumentation signal.

6. The method of claim 5 wherein step a) further comprises the step of:

i) inserting a unique variable assignment statement into the source code, wherein the variable assignment statement is adjacent to at least one associated sequential statement; and

ii) inserting a unique output signal assignment statement into the source code, wherein the unique output signal is assigned a value associated with the unique variable.

7. The method of claim 6 wherein the variable is assigned a first value in step a)i), the method further comprising the step of:

iii) modifying the source code to initialize the unique variable to a second value.

8. The method of claim 5 wherein step a) is repeated to create a unique instrumented output signal for each list of sequential statements in the source code, wherein each list corresponds to a synthesizable executable branch of the source code.

9. The method of claim 5 further comprising the step of:

c) generating cross-reference instrumentation data mapping each statement in a selected list to the instrumented output signal associated with that list for every list in the source code.

10. The method of claim 9 further comprising the steps of:

d) simulating the gate level design using at least one of the instrumentation signals to establish a simulation breakpoint.

11. The method of claim 5 further comprising the steps of:

c) displaying the source code, wherein at least one statement within a selected list is highlighted if the instrumentation signal corresponding to the selected list changes to a pre-determined value.

12. A method of generating a gate-level netlist, comprising the steps of:

a) receiving register transfer level (RTL) synthesizable source code including synthesizable statements;

b) inserting a unique local variable assignment statement into the source code for each branch of code having a list of at least one sequential statement, wherein the unique local variable assignment statement is adjacent to at least one statement within the list;

c) inserting a corresponding instrumentation signal assignment statement into the source code for each of the inserted local variables, wherein the instrumentation signal is assigned a value of the corresponding unique local variable; and

d) synthesizing the source code into a gate-level design including the instrumentation signals.

13. The method of claim 12 wherein step b) further comprises the steps of:

i) assigning each unique local variable a first value; and

ii) initializing each local variable with second value.

14. The method of claim 12 further comprising the step of

e) mapping every statement within a selected list to the corresponding instrumentation signal for that selected list as cross-reference instrumentation data.

15. The method of claim 12 further comprising the steps of:

e) setting a breakpoint at a selected statement of the source code;

f) identifying the instrumentation signal corresponding to the list associated with the selected statement as a breakpoint signal; and

g) simulating the gate-level design, wherein simulation is halted at a simulation cycle that results in the breakpoint signal transitioning to a pre-determined value.

16. A method of generating a gate level netlist, comprising the steps of:

a) receiving register transfer level (RTL) synthesizable source code including synthesizable statements;

b) modifying the source code to generate a corresponding sampled version of each signal event in a selected process;

c) modifying the source code to duplicate the selected process;

d) replacing each occurrence of a selected signal event with the corresponding sampled version in the duplicated process;

e) replacing each list of sequential statements within an executable branch of the duplicated process with a unique variable assignment statement;

f) modifying the duplicated process to include an instrumentation signal assignment for each unique variable; and

g) synthesizing the modified source code into a gate-level design.

17. The method of claim 16 wherein step e) further comprises the steps of:

i) assigning the unique variables a first value; and

ii) initializing the unique variables with second value.

18. The method of claim 16 further comprising the step of

e) mapping every statement within each selected list to its corresponding instrumentation signal.

19. The method of claim 16 further comprising the steps of:

h) setting a breakpoint at a selected statement of the source code;

i) identifying the instrumentation signal corresponding to the list associated with the selected statement as a breakpoint signal; and

j) simulating the gate-level design, wherein simulation is halted at a simulation cycle that results in a transition of the breakpoint signal to a predetermined value.

20. A method of debugging a gate-level design including the steps of:

a) setting a breakpoint at a selected statement of a register transfer level (RTL) synthesizable source code;

b) inserting a local variable assignment statement adjacent to at least one statement in a list of sequential statements, wherein the list corresponds to an executable branch of the source code including the selected statement;

c) modifying the source code to include an instrumentation signal assignment statement for the local variable; and

d) generating a gate-level design from the modified source code.

21. The method of claim **20** further comprising the steps of:

e) simulating the gate-level design, wherein simulation is halted at a simulation cycle that results in a transition of the instrumentation signal to a pre-determined value.

22. The method of claim **20** wherein step b) further comprises the steps of:

i) assigning the local variable a first value; and

ii) initializing the local variable with second value.

23. The method of claim **20** further comprising the step of

e) mapping every statement within the executable branch of source code to the instrumentation signal.

24. A method of simulating a gate-level design comprising the steps of:

a) identifying a sensitivity list of a process;

b) generating logic to identify signal events for any signal in the sensitivity list; and

c) identifying the process as active during simulation when a signal event occurs for any signal in the sensitivity list.

25. The method of claim **24** wherein step c) further comprises the step of:

i) highlighting a source code description of the process displayed during simulation.

26. The method of claim **24** wherein step b) further comprises the step of:

i) sampling each signal in the sensitivity list to generate corresponding instrumented signals; and

ii) comparing each signal in the sensitivity list with its corresponding instrumented signal to test each signal in the sensitivity list for an event.

27. The method of claim **26** wherein step c) further comprises the step of:

i) generating an active process output signal defined by logically ORing the results of the comparisons.

28. A storage medium having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code, wherein when executed the instructions enable the processor to synthesize the source code into a gate-level netlist including at least one instrumentation signal, wherein the instrumentation signal is indicative of an execution status of at least one synthesizable statement of the source code.

29. The storage medium of claim **28** wherein the processor performs the steps of:

i) inserting a unique variable assignment statement into the source code, wherein the variable assignment statement is adjacent to at least one associated sequential statement; and

ii) inserting a unique output signal assignment statement into the source code, wherein the unique output signal is assigned a value associated with the unique variable.

30. A storage medium having stored therein processor executable instructions for generating a gate-level design from a register transfer level (RTL) synthesizable source code, wherein when executed the instructions enable the processor to perform the steps of:

a) inserting a unique local variable assignment statement into the source code for each branch of code having a list of at least one sequential statement, wherein the unique local variable assignment statement is adjacent to at least one statement within the list;

b) inserting a corresponding instrumentation signal assignment statement into the source code for each of the inserted local variables, wherein the instrumentation signal is assigned a value of the corresponding unique local variable; and

c) synthesizing the source code into a gate-level design including the instrumentation signals.

31. The storage medium of claim **30** having stored therein further instructions to enable the processor to perform the step of:

d) mapping every statement within each selected list to its corresponding instrumentation signal.

32. A storage medium having stored therein processor executable instructions for debugging a gate level design during simulation, wherein when a breakpoint is set at a selected statement of a register transfer level (RTL) synthesizable source code the instructions enable the processor to perform the steps of:

a) inserting a local variable assignment statement adjacent to at least one statement in a list of sequential statements within the source code, wherein the list corresponds to an executable branch of the source code including the selected statement;

b) modifying the source code to include an instrumentation output signal assignment statement for the local variable; and

c) generating a gate-level design from the modified source code.

33. The storage medium of claim **32** having stored therein further instructions to enable the processor to perform the step of:

d) mapping every statement within each selected list to its corresponding instrumentation signal.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,240,376 B1
DATED : May 29, 2001
INVENTOR(S) : Alain Raynaud and Luc M. Burgun

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 1,
Line 11, "hen" should read -- when --.
Line 17, "modem" should read -- modern --.
Line 37, "simulator. alternatively," should read -- simulator. Alternatively, --.
Line 38, "converting the ate-" should read -- converting the gate- --.
Line 47, "RTh" should read -- RTL --.

Column 2,
Line 39, "RTh" should read -- RTL --.

Column 10,
Line 33, "signal CLX_" should read -- signal CLK_ --.

Signed and Sealed this

Sixteenth Day of July, 2002

Attest:

Attesting Officer

JAMES E. ROGAN
Director of the United States Patent and Trademark Office

U.S. Patent No. 6,132,109

Dated October 17, 2000

US006132109A

United States Patent [19]

Gregory et al.

[11] Patent Number: 6,132,109

[45] Date of Patent: *Oct. 17, 2000

[54] **ARCHITECTURE AND METHODS FOR A HARDWARE DESCRIPTION LANGUAGE SOURCE LEVEL DEBUGGING SYSTEM**

[75] Inventors: **Brent Gregory**; **Trinanjan Chatterjee**; **Jing C. Lin**; **Srinivas Raghvendra**, all of Sunnyvale; **Emil Girczyc**, Los Altos; **Paul Estrada**, Mountain View; **Andrew Seawright**, Cupertino, all of Calif.

[73] Assignee: **Synopsys, Inc.**, Mountain View, Calif.

[*] Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

[21] Appl. No.: **08/253,470**

[22] Filed: **Jun. 3, 1994**

Related U.S. Application Data

[63] Continuation-in-part of application No. 08/226,147, Apr. 12, 1994, abandoned.

[51] **Int. Cl.**[7] **G06F 9/445**; G06F 9/45; G06F 15/00

[52] **U.S. Cl.** **395/704**; 364/489

[58] **Field of Search** 395/700, 701, 395/704, 705; 364/578, 488, 489, 490, 491

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,546,435	10/1985	Herbert et al.	364/300
4,667,290	5/1987	Goss et al.	395/707
4,703,435	10/1987	Darringer et al.	364/489
4,827,427	5/1989	Hyduke	364/489
4,852,173	7/1989	Bahl et al.	381/43
4,866,663	9/1989	Griffin	395/500
4,868,770	9/1989	Smith et al.	364/578
4,882,690	11/1989	Shinsha et al.	364/490

(List continued on next page.)

OTHER PUBLICATIONS

Weiss, Ray, "ASIC's forcing logic synthesis' hand," Electronic Engineering Times, n516, T16 (pp. 1–2), Dec. 12, 1988.

Weiss, Ray, "Designers moving toward high–level logic representation via logic synthesis—Logic synthesis edging up design hierarchy," Electronic Engineering Times, n526, 81, (pp. 1–10), Feb. 6, 1989.

Weiss, Ray, "Synopsys fine–tunes logic synthesis," Electronic Engineering Times, n534, 138 (pp. 1–3), Apr. 17, 1989.

Brian Ebert et al., "SeeSaw: A Verilog Synthesis Viewer," 2nd Annual International Verilog HDL Conference, Design Excellence for Today and Tomorrow; Santa Clara, CA, Mar. 22–24, 1993, pp. 55–60.

Lis et al., "VHDL Synthesis Using Structured Modeling," 26th ACM/IEEE Design Automation Conference, Jun. 25, 1989, pp. 606–609.

Sougata Mukherjea, et al., "Applying Algorithm Animation Techniques for Program Tracing, Debugging, and Understanding," Proceedings 15th International Conference on Software Engineering, May 17, 1993, Baltimore, MD, pp. 456–465.

Design Analyzer Reference (per paper #9), "Text Viewer," v.3.1, pp. 1–15, Mar. 1994.

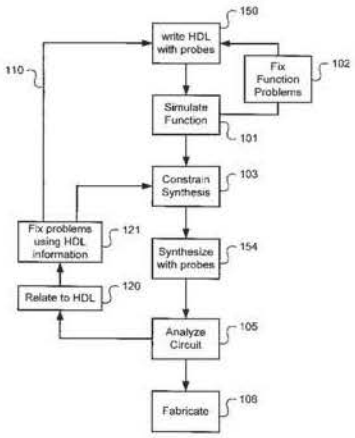
(List continued on next page.)

Primary Examiner—Tariq R. Hafiz
Assistant Examiner—Michael Pender
Attorney, Agent, or Firm—Brown Raysman Millstein Felder & Steiner LLP; Jonathan T. Kaplan

[57] **ABSTRACT**

This invention provides a method for displaying circuit analysis results corresponding to parts of the circuit near the portion of the hardware description language (HDL) specification that generated that part of the circuit. The invention also includes a method for using probe statements in the HDL specification to mark additional points in the initial circuit that should not be eliminated during optimization. This improves the ability to display circuit analysis results near the appropriate part of the HDL specification.

2 Claims, 33 Drawing Sheets



A1103

U.S. PATENT DOCUMENTS

4,907,180	3/1990	Smith	364/578
4,942,536	7/1990	Watanabe et al.	364/490
4,942,615	7/1990	Hirose	364/578
4,967,386	10/1990	Maeda et al.	364/578
4,970,664	11/1990	Kaiser et al.	364/521
5,111,413	5/1992	Lazensky et al.	364/578
5,146,583	9/1992	Matsunaka et al.	395/500
5,191,541	3/1993	Landman et al.	364/489
5,191,646	3/1993	Naito et al.	395/161
5,265,254	11/1993	Blasciak et al.	395/700
5,282,146	1/1994	Aihara et al.	364/489
5,282,148	1/1994	Poirot et al.	364/491
5,329,471	7/1994	Swoboda et al.	364/578
5,335,191	8/1994	Kundert et al.	364/578
5,377,997	1/1995	Wilden et al.	273/434
5,437,037	7/1995	Furuichi	395/700
5,446,900	8/1995	Kimelman	395/700
5,452,239	9/1995	Dai et al.	364/578
5,493,507	2/1996	Shinde et al.	395/500.35
5,530,841	6/1996	Gregory et al.	395/500
5,541,849	7/1996	Rostoker et al.	364/489
5,544,066	8/1996	Rostoker et al.	364/489
5,544,067	8/1996	Rostoker et al.	364/489
5,544,068	8/1996	Takimoto et al.	364/489
5,553,002	9/1996	Dangelo et al.	364/489
5,555,201	9/1996	Dangelo et al.	364/489
5,557,531	9/1996	Rostoker et al.	364/489
5,581,738	12/1996	Dambrowski	395/500.17

OTHER PUBLICATIONS

D. E. Thomas et al., "Algorithmic and Register–Transfer Level Synthesis: The System Architect's Workbench," pp. 257–274, publication date unknown.

Pure Software Inc., Pure Coverage Data Sheet (Web Page), copyright 1996.

Pure Software Inc., Quantify Data Sheet (Web Page), copyright 1996.

Pure Software Inc., Purify, Finding Run–Time Memory Errors (Web Page), coyright 1996.

Reed Hastings et al., Pure Software, Inc., Purify, Usenix White Paper on Purify (Web Page), copyright 1996.

Printout of Web Page for Centerline, Code Center, Release 4, Nov. 1994.

HDC Computer Corporation, FirstApps User's Guide, pp. 112–116, copyright 1992.

Louis Trevillyan, "An Overview of Logic Synthesis Systems," 1987, pp. 166–172.

Timothy Kam, "Comparing Layouts with HDL Models: A Formal Verification Technique," 1992, pp. 588–591.

"A Programming the User Interface", Manual of Symbolics, Inc., 4 New England Tech Center, 555 Virginia Road, Concord, MA 01742, title page, pp. iii–v and pp. 1–134, Sep. 1986.

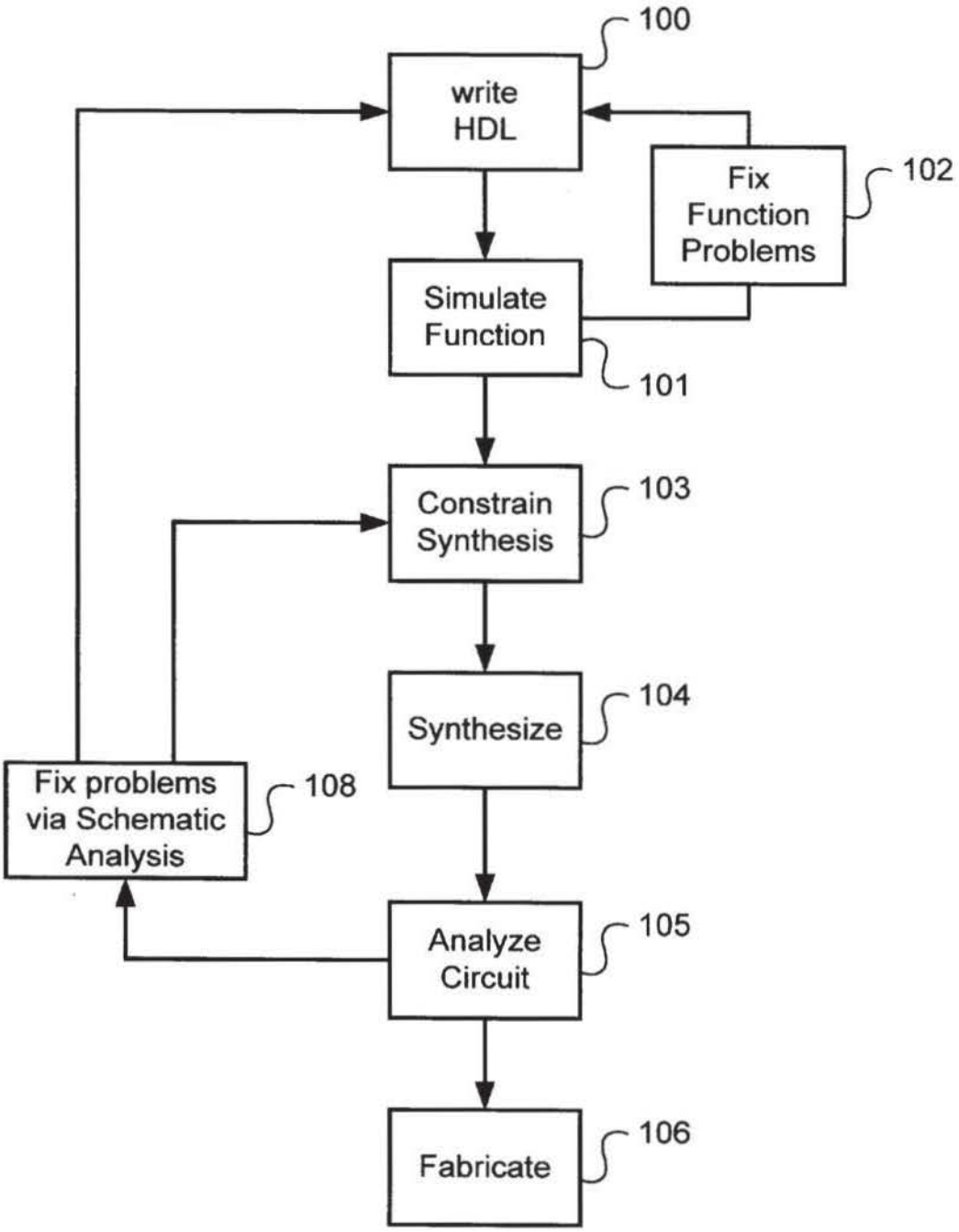

100
write HDL
Fix Function Problems
102
Simulate Function
101
Constrain Synthesis
103
Synthesize
104
Fix problems via Schematic Analysis
108
Analyze Circuit
105
Fabricate
106

Figure 1

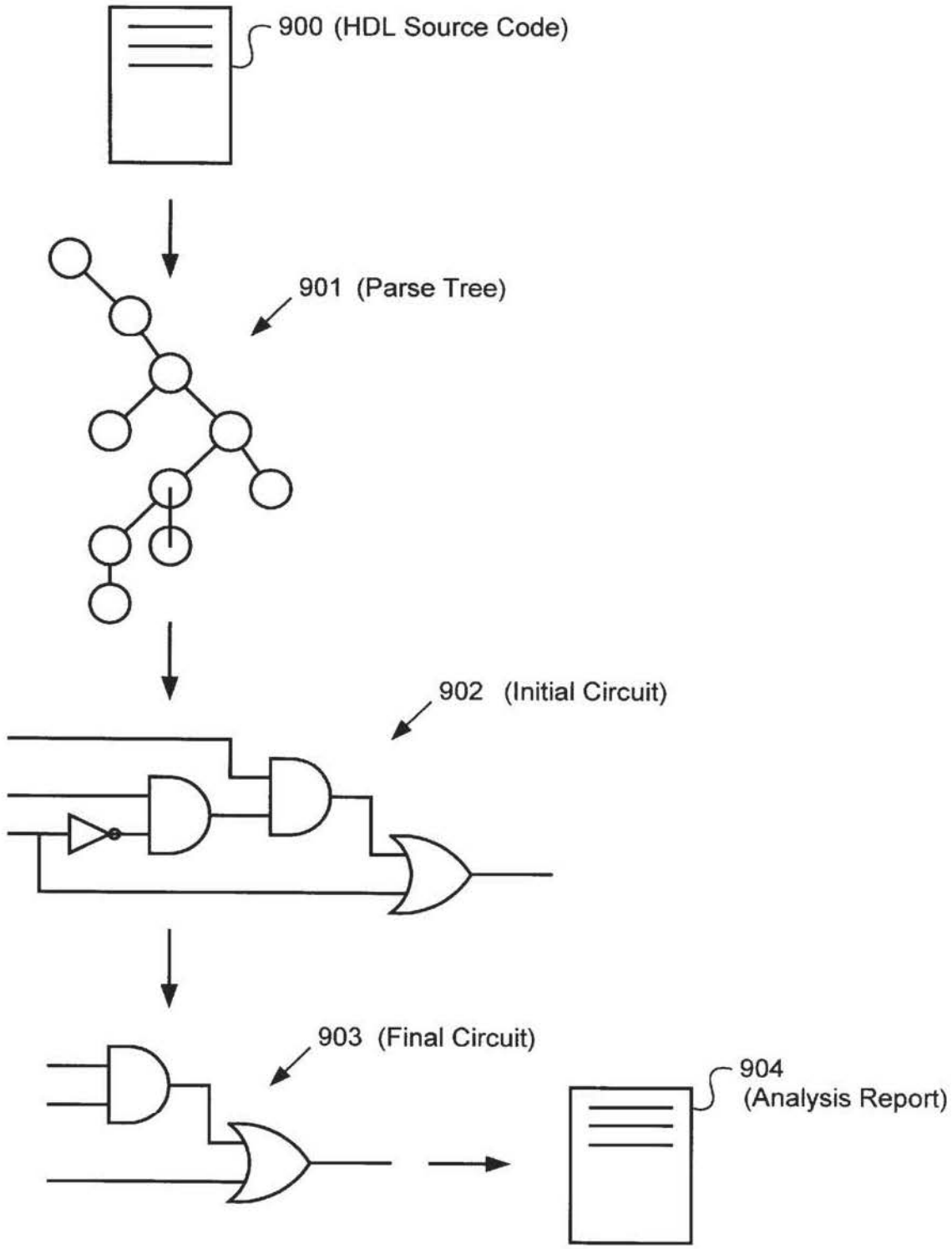

900 (HDL Source Code)
901 (Parse Tree)
902 (Initial Circuit)
903 (Final Circuit)
904
(Analysis Report)

Figure 2

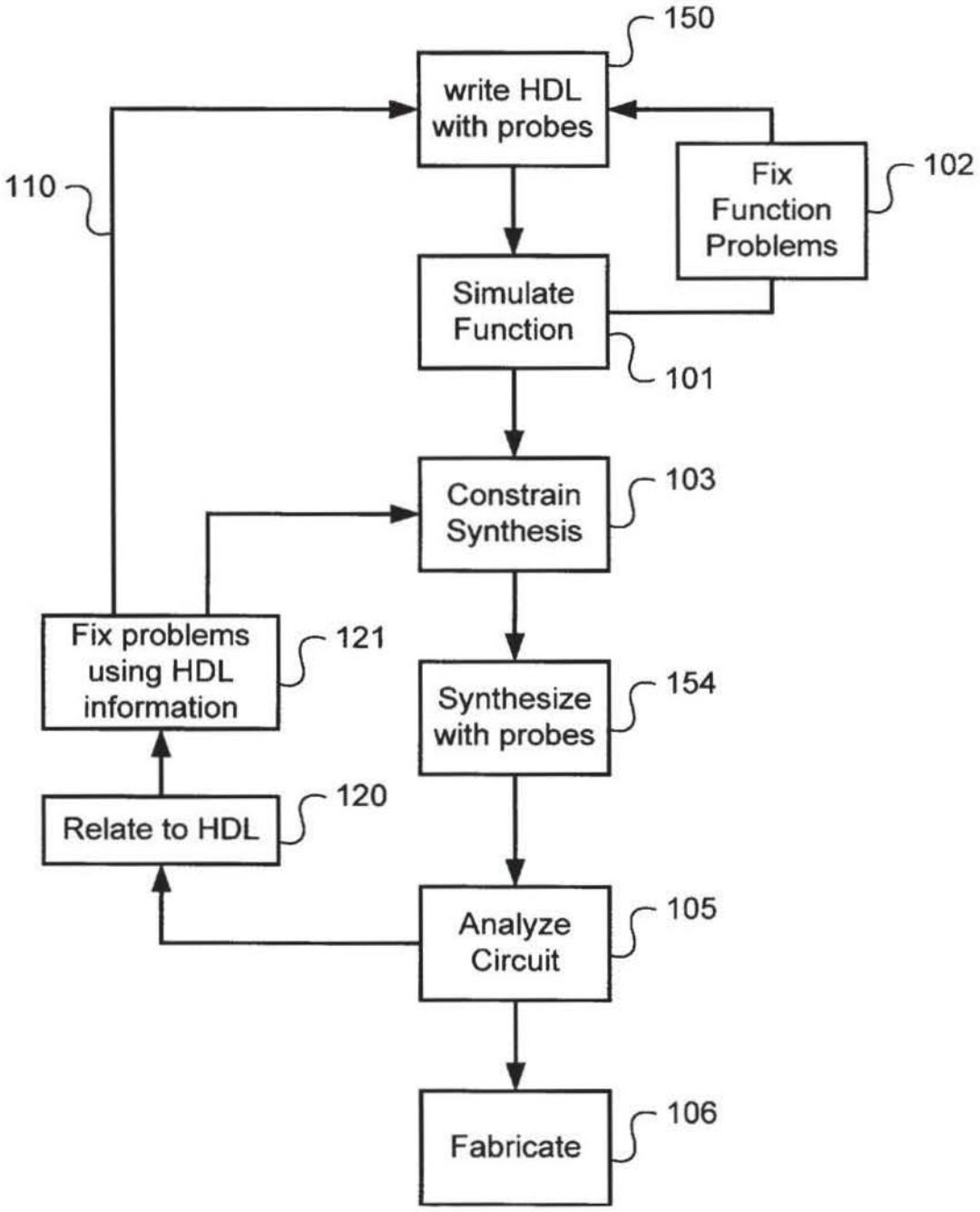

150
write HDL with probes
110
Fix Function Problems
102
Simulate Function
101
Constrain Synthesis
103
Fix problems using HDL information
121
Synthesize with probes
154
Relate to HDL
120
Analyze Circuit
105
Fabricate
106

Figure 3

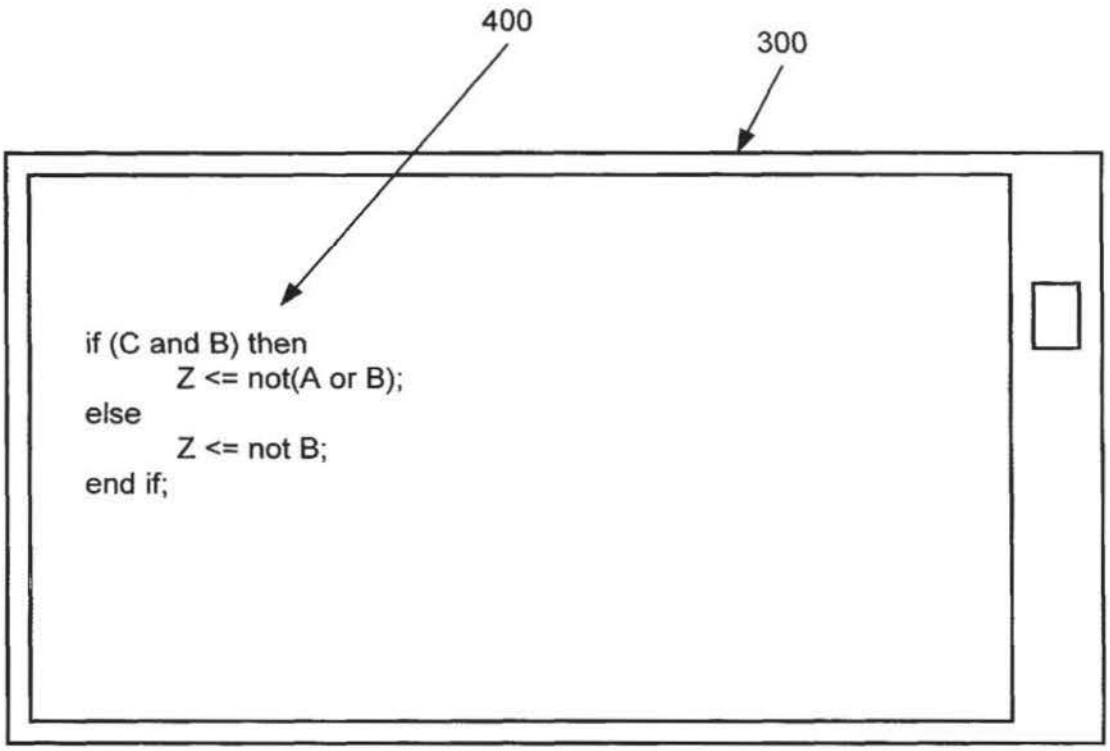


Figure 4

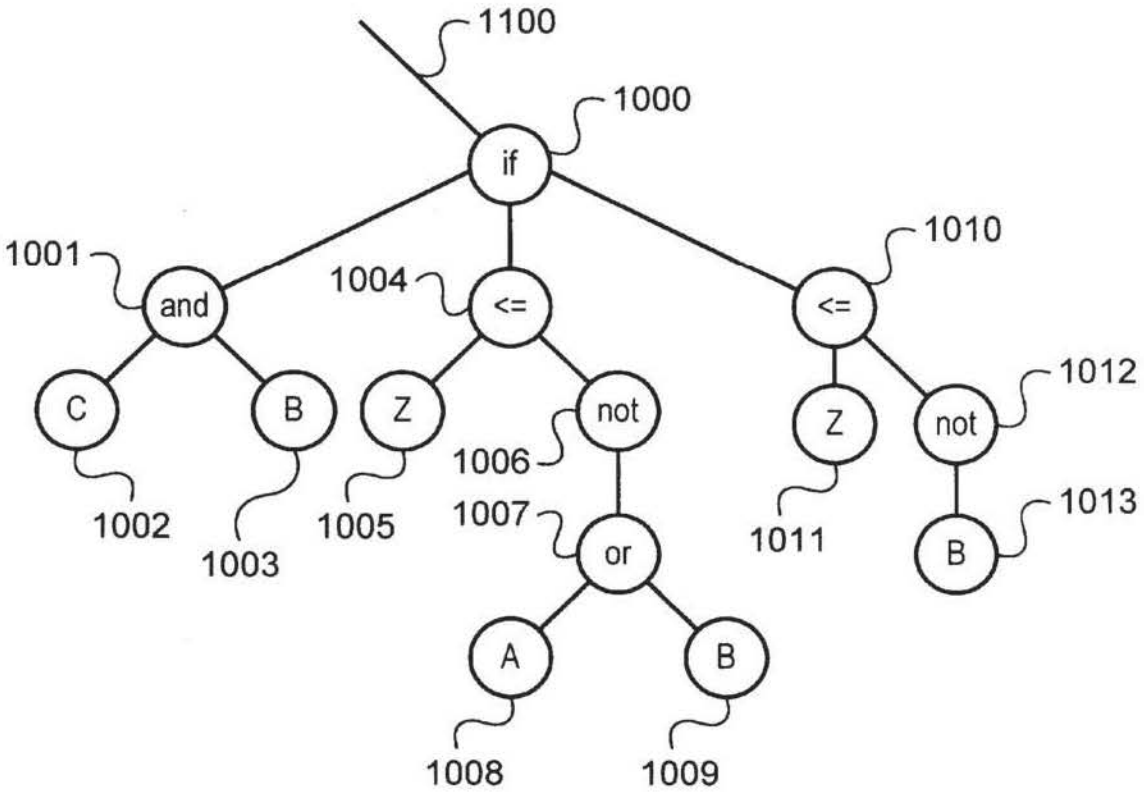

1100
1000
if
1001
and
1004
<=
1010
<=
C
B
Z
not
Z
not
1012
1006
1002
1003
1005
1007
1011
1013
or
B
A
B
1008
1009

Figure 5

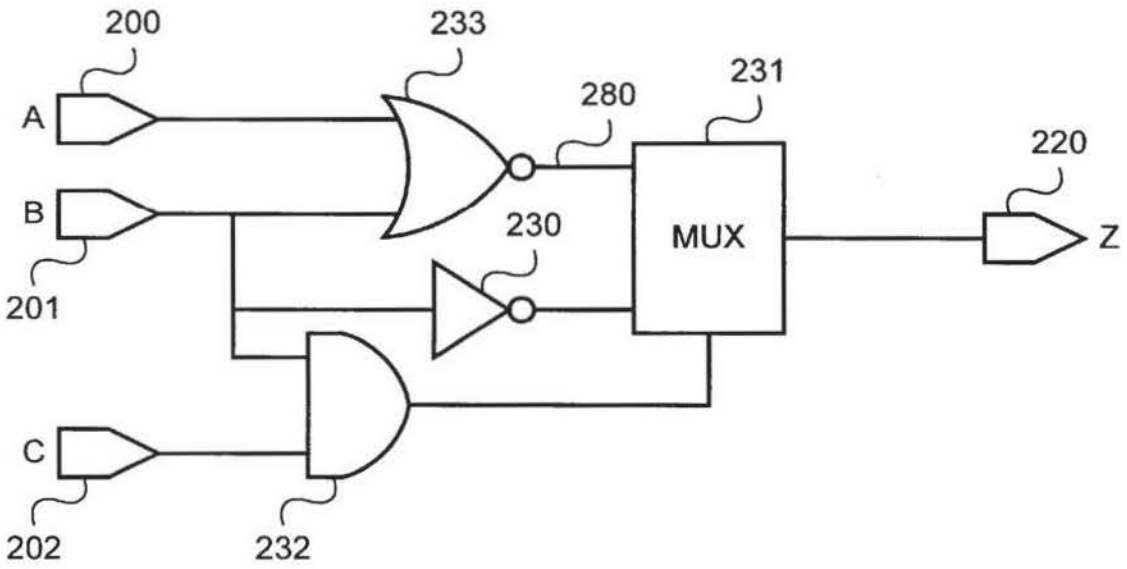


Figure 6

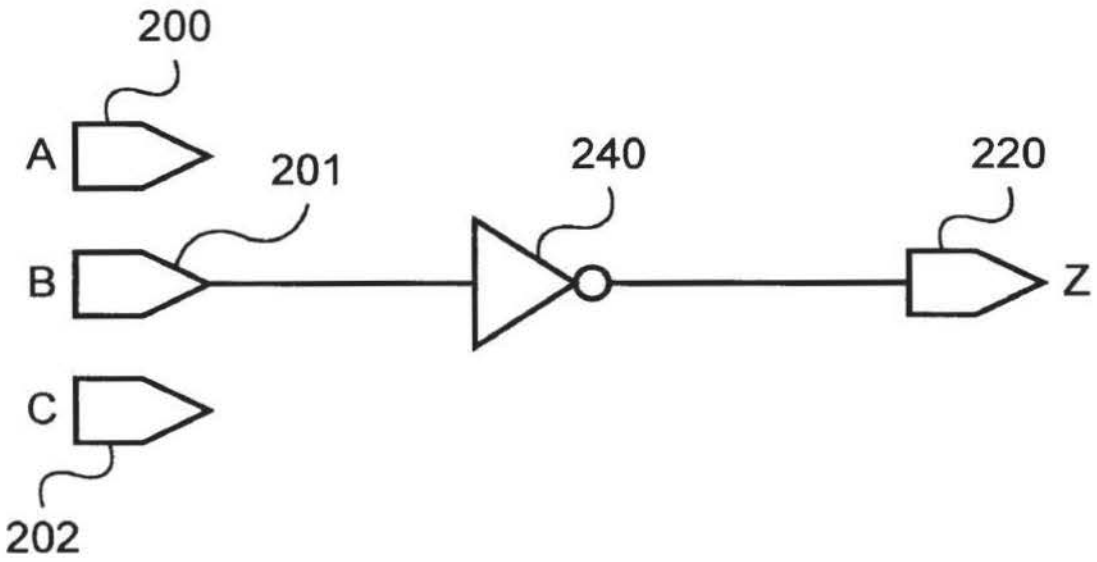


Figure 7

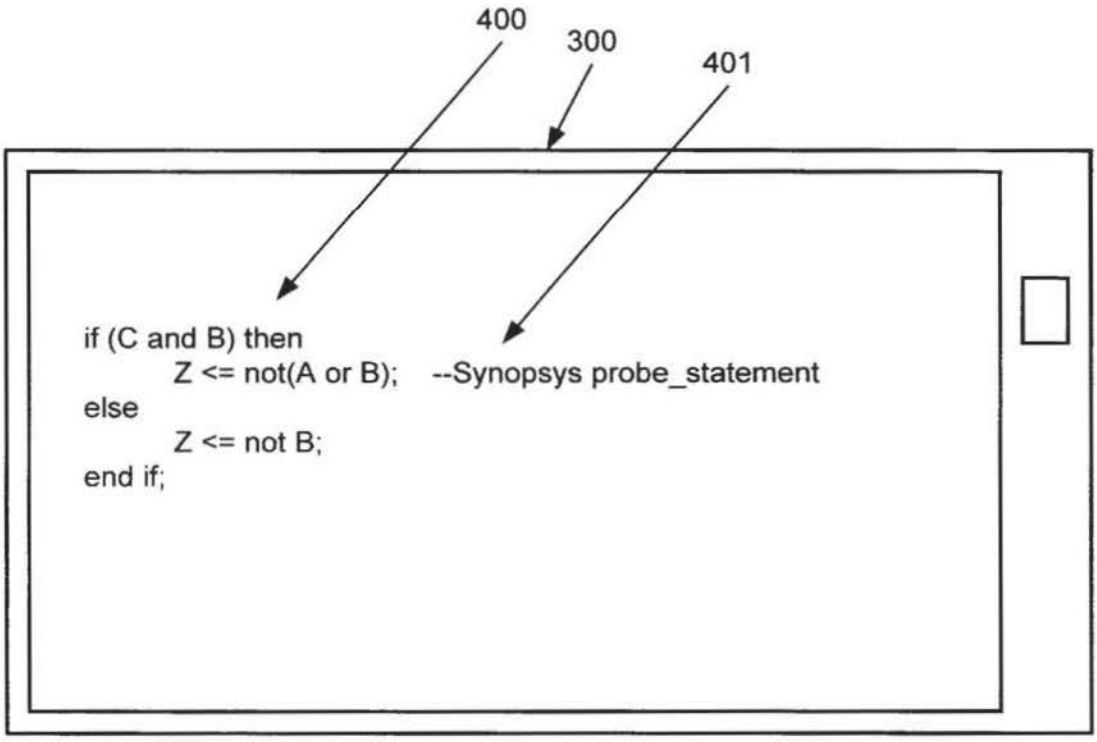

400
300
401
if (C and B) then
Z <= not(A or B); --Synopsys probe_statement
else
Z <= not B;
end if;

Figure 8

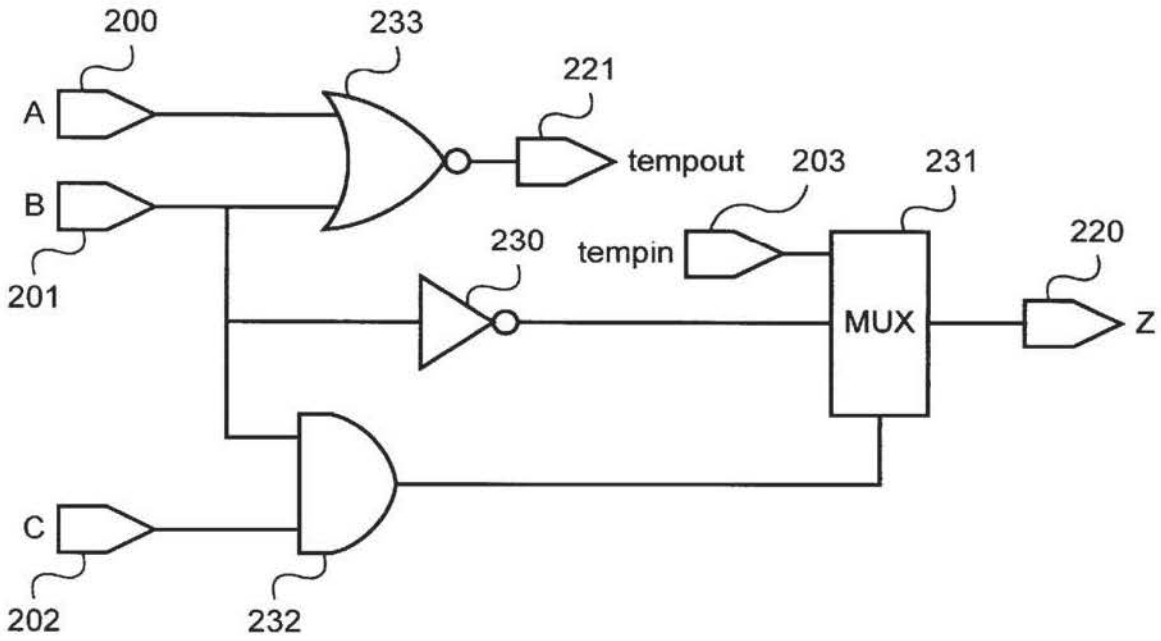

200
233
221
A
B
tempout
203
231
230
tempin
220
201
MUX
Z
C
202
232

Figure 9

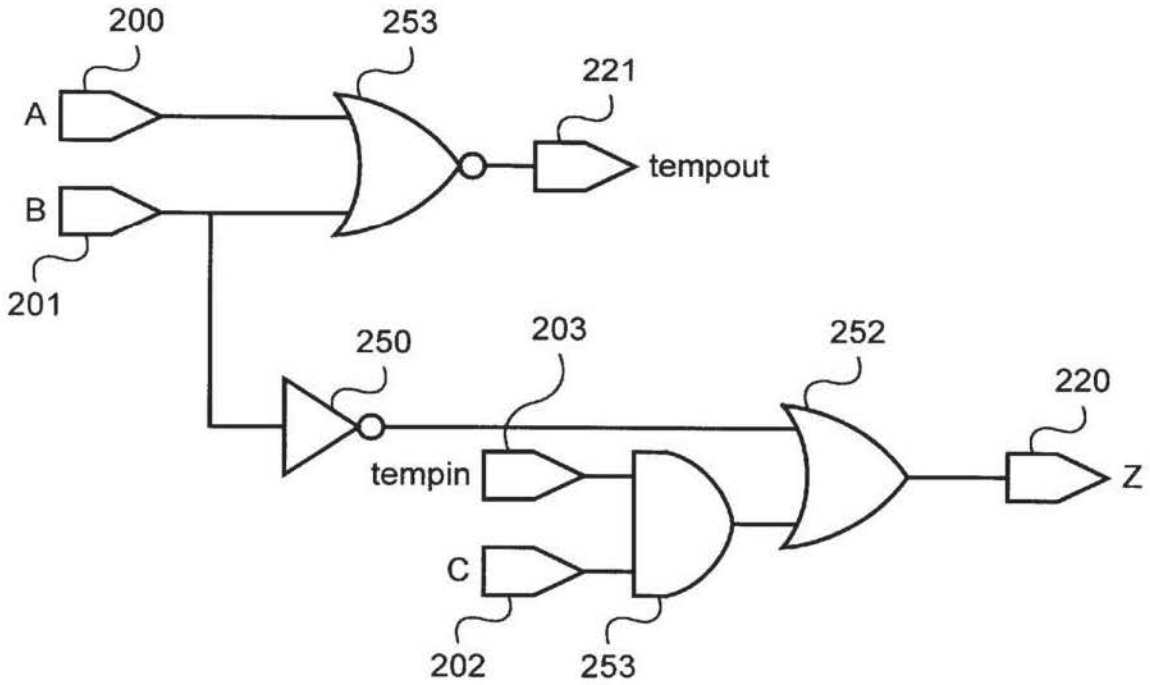


Figure 10

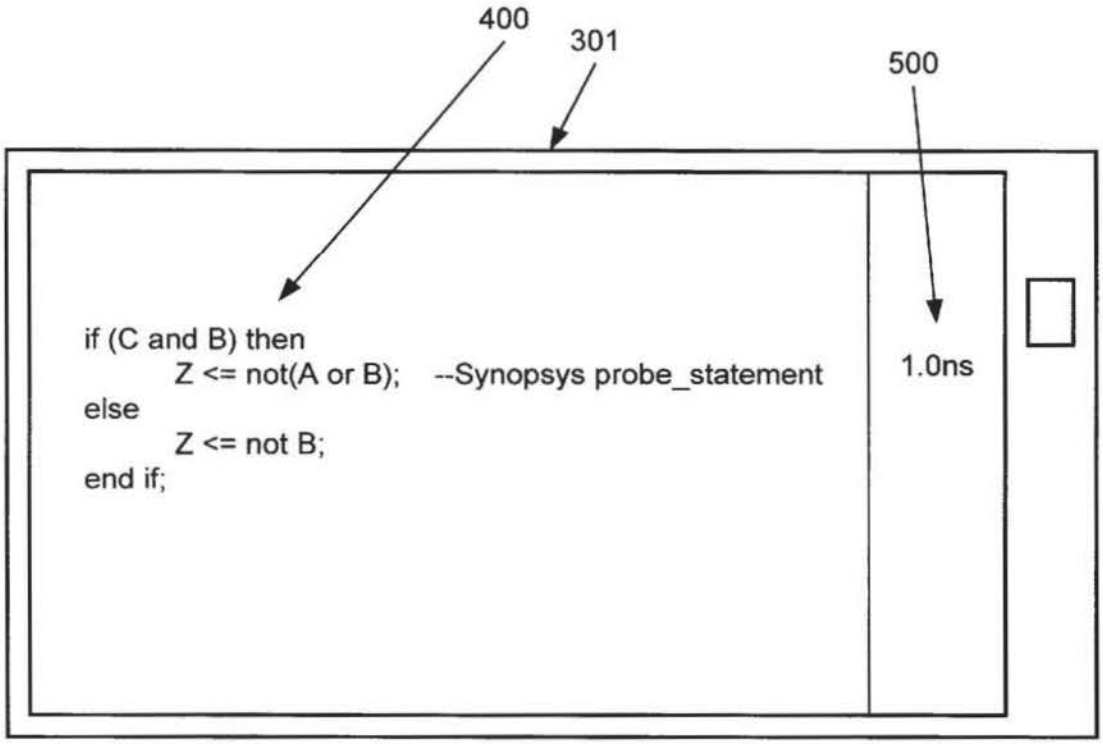

400
301
500
if (C and B) then
Z <= not(A or B); --Synopsys probe_statement
else
Z <= not B;
end if;
1.0ns

Figure 11

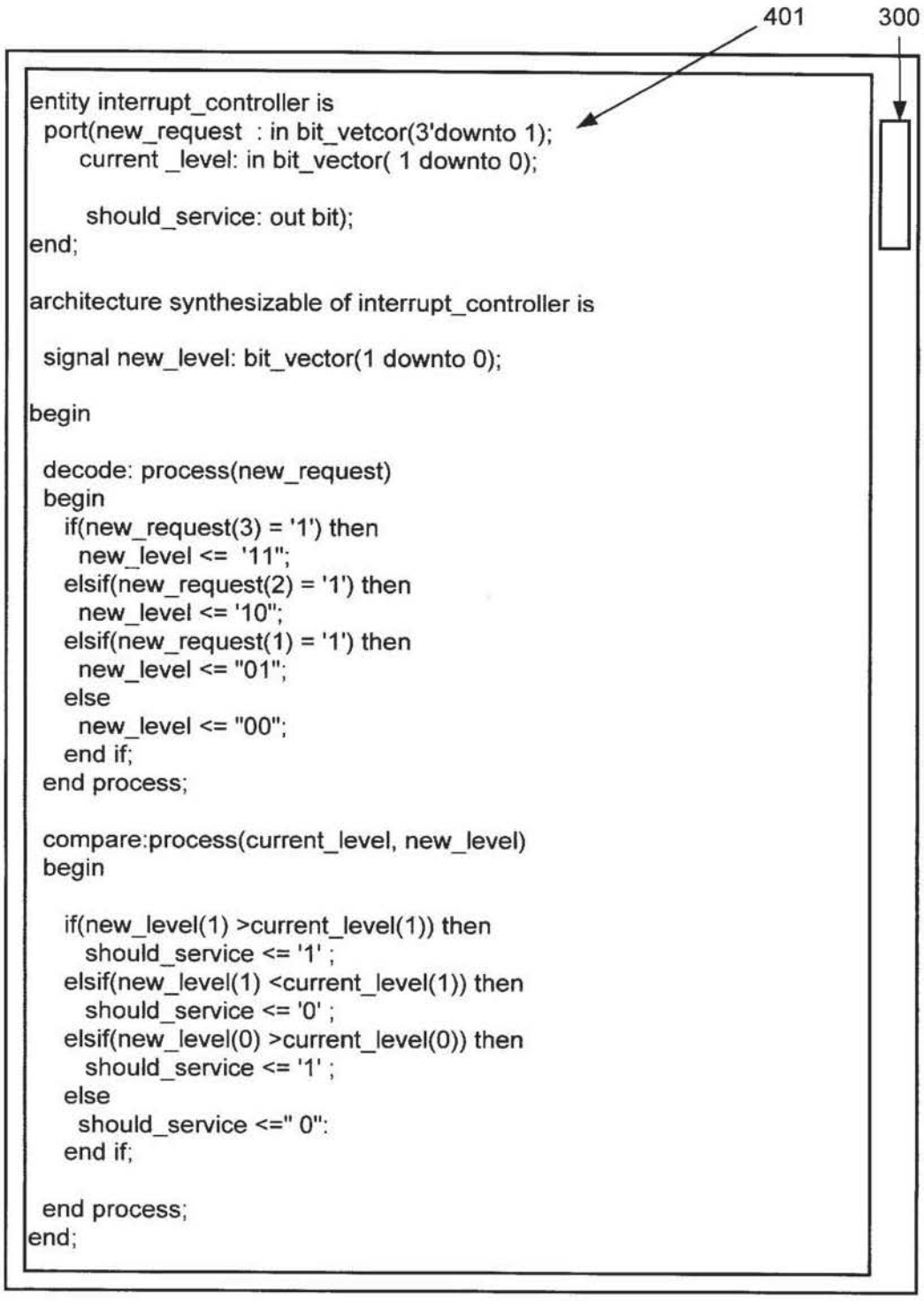
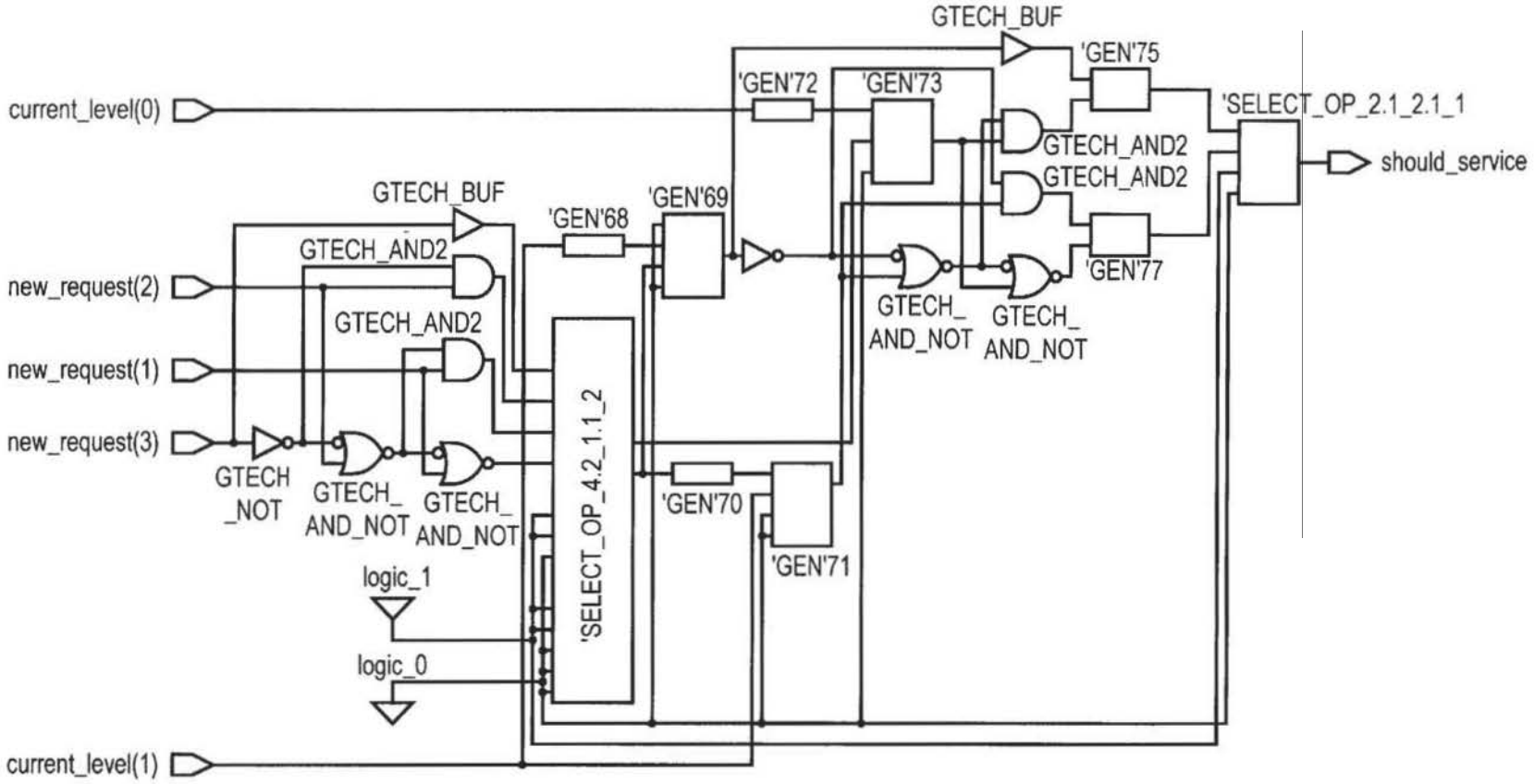

401
300
entity interrupt_controller is
port(new_request : in bit_vetcor(3'downto 1);
current _level: in bit_vector(1 downto 0);
should_service: out bit);
end;
architecture synthesizable of interrupt_controller is
signal new_level: bit_vector(1 downto 0);
begin
decode: process(new_request)
begin
if(new_request(3) = '1') then
new_level <= '11";
elsif(new_request(2) = '1') then
new_level <= '10";
elsif(new_request(1) = '1') then
new_level <= "01";
else
new_level <= "00";
end if;
end process;
compare:process(current_level, new_level)
begin
if(new_level(1) >current_level(1)) then
should_service <= '1' ;
elsif(new_level(1) <current_level(1)) then
should_service <= '0' ;
elsif(new_level(0) >current_level(0)) then
should_service <= '1' ;
else
should_service <=" 0":
end if;
end process;
end;

Figure 12



design : interrupt_controller	designer :	date : 4/5/94
technology : lsi_10k	company :	sheet : 1 of 1

Figure 13

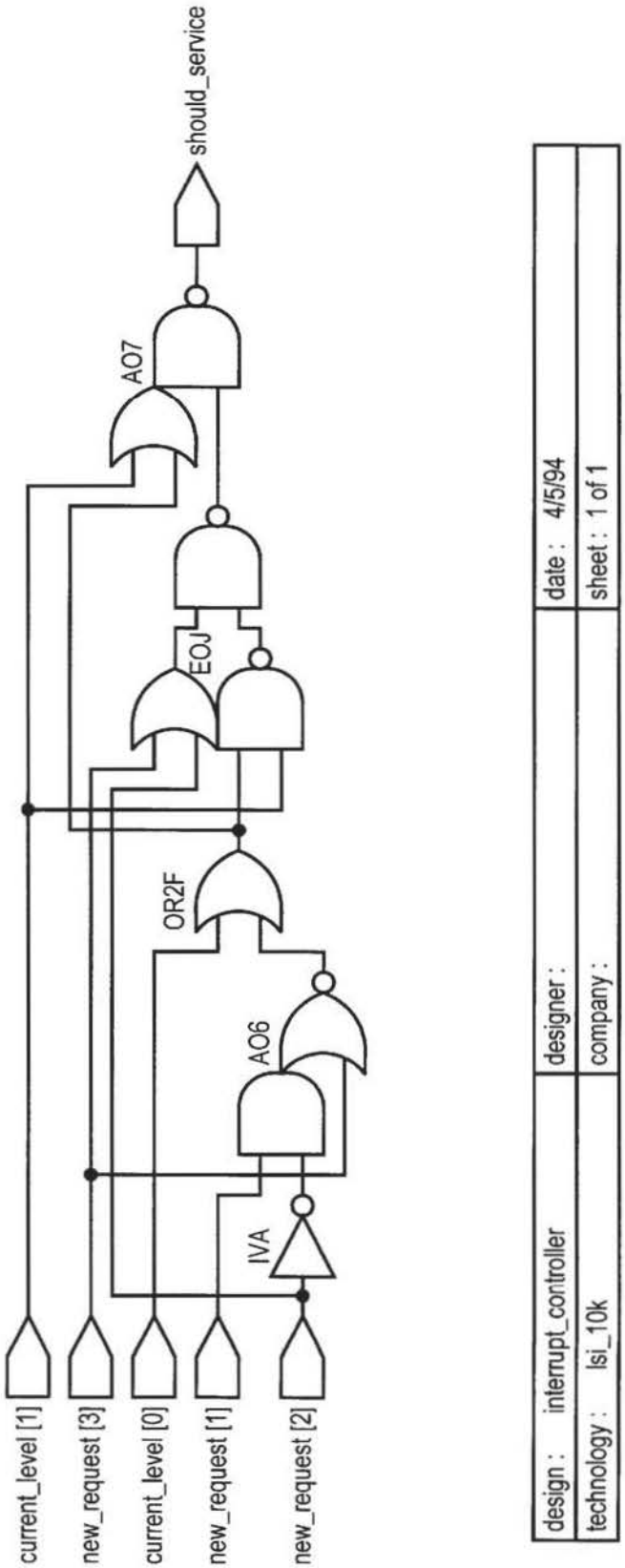
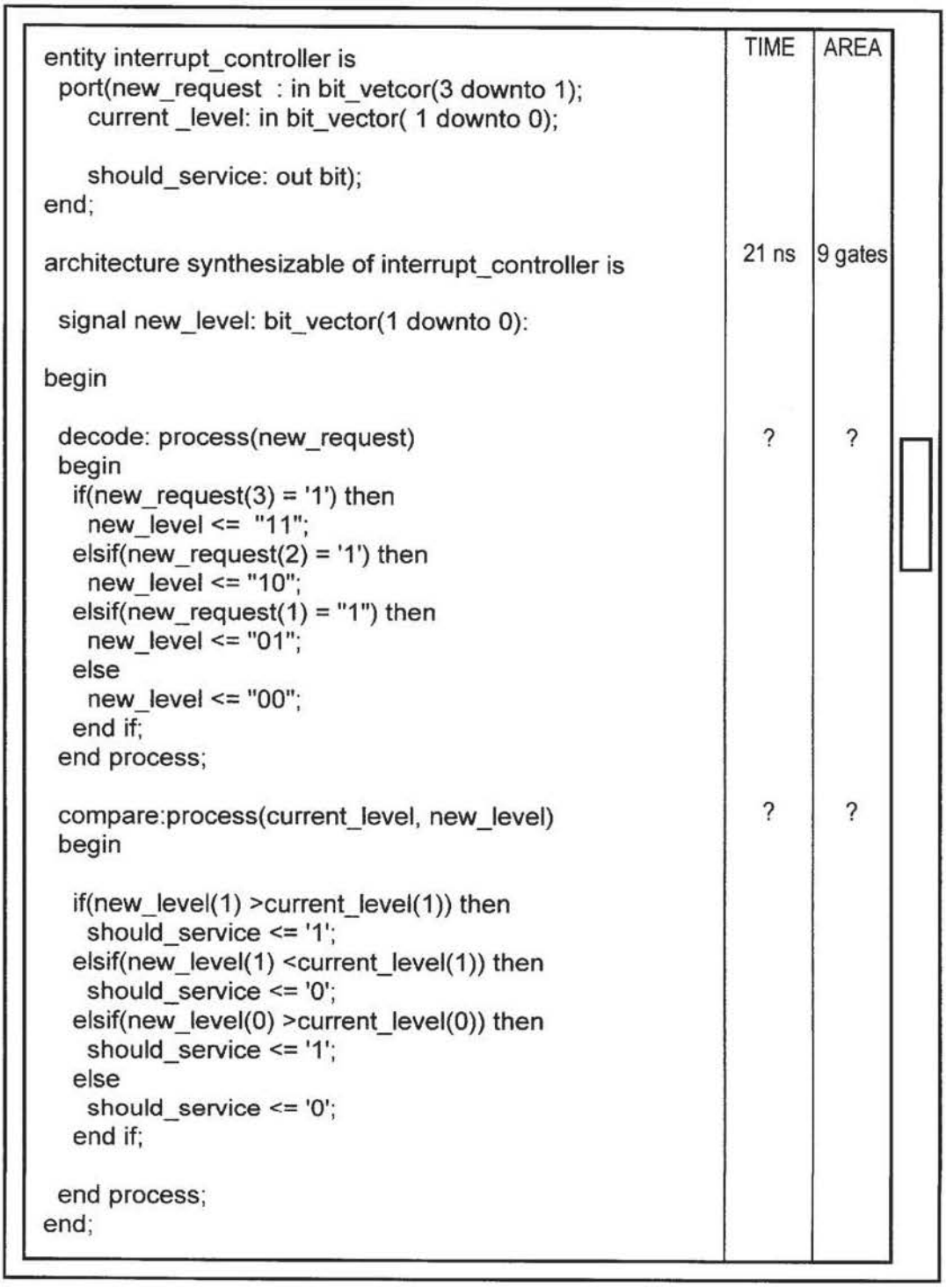


Figure 14



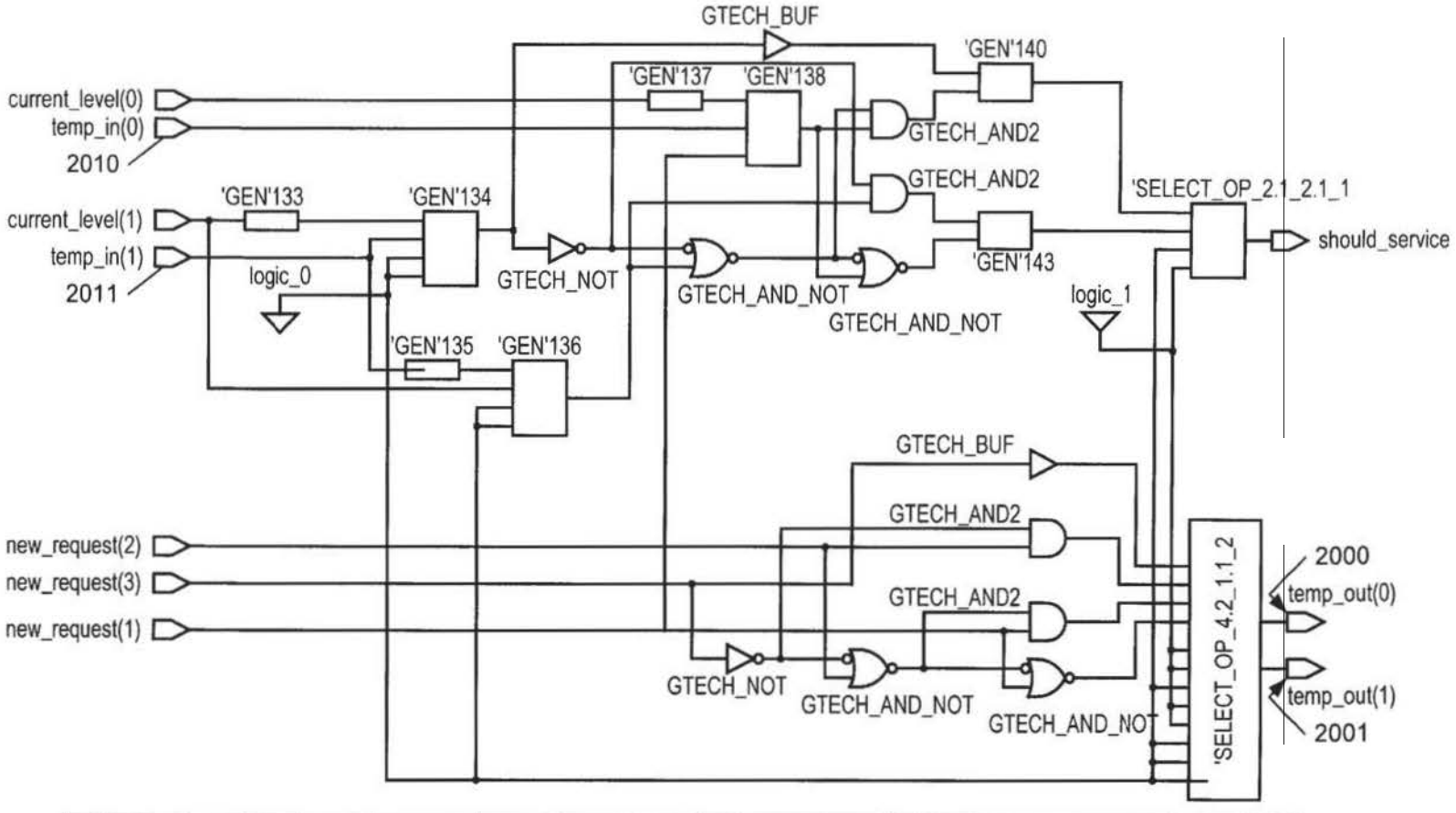
```
entity interrupt_controller is
  port(new_request  : in bit_vetcor(3 downto 1);
     current _level: in bit_vector( 1 downto 0);

     should_service: out bit);
end;

architecture synthesizable of interrupt_controller is

  signal new_level: bit_vector(1 downto 0):

begin

  decode: process(new_request)
  begin
    if(new_request(3) = '1') then
      new_level <=  "11";
    elsif(new_request(2) = '1') then
      new_level <= "10";
    elsif(new_request(1) = "1") then
      new_level <= "01";
    else
      new_level <= "00";
    end if;
  end process;

  compare:process(current_level, new_level)
  begin

    if(new_level(1) >current_level(1)) then
      should_service <= '1';
    elsif(new_level(1) <current_level(1)) then
      should_service <= '0';
    elsif(new_level(0) >current_level(0)) then
      should_service <= '1';
    else
      should_service <= '0';
    end if;

  end process;
end;
```

TIME	AREA
21 ns	9 gates
?	?
?	?

Figure 15

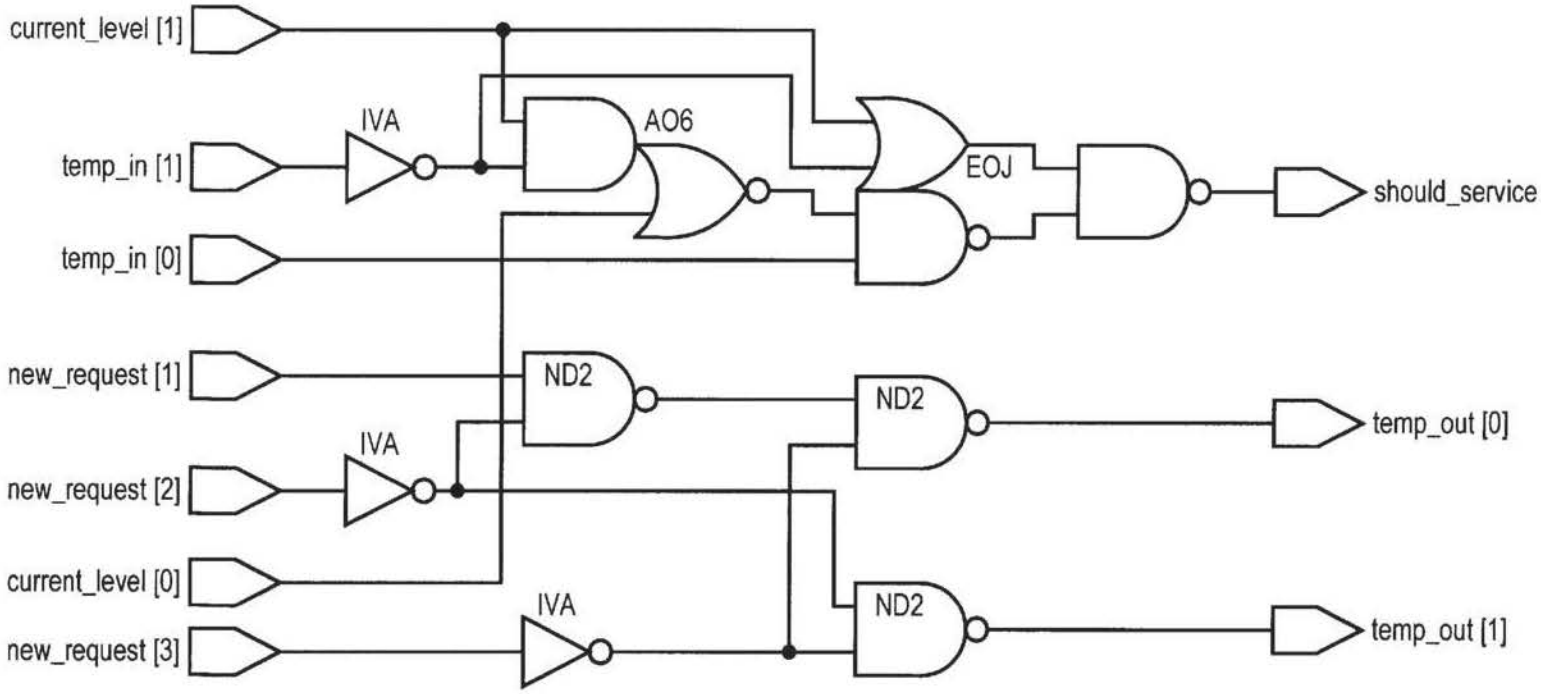
```
entity interrupt_controller is
  port(new_request  : in bit_vector(3 downto 1);
       current_level: in bit_vector(1 downto 0);

       should_service: out bit);
end;

architecture synthesizable of interrupt_controller is

  signal new_level: bit_vector(1 downto 0);

begin
  --Synopsys block_probe_begin
  decode: process(new_request)
  begin
    if(new_request(3) = '1') then
      new_level <= "11";
    elsif(new_request(2) = '1') then
      new_level <= "10";
    elsif(new_request(1) = '1') then
      new_level <= "01";
    else
      new_level <= "00";
    end if;
  end process:
  --Synopsys block_probe_end

  compare: process(current_level,new_level)
  begin

    if(new_level(1) > current_level(1)) then
      should_service <= '1';
    elsif(new_level(1) < current_level(1)) then
      should_service <= '0';
    elsif(new_level(0) > current_level(0)) then
      should_service <= '1';
    else
       should_service <= '0';
    end if;
  end process;
end;
```

Figure 16



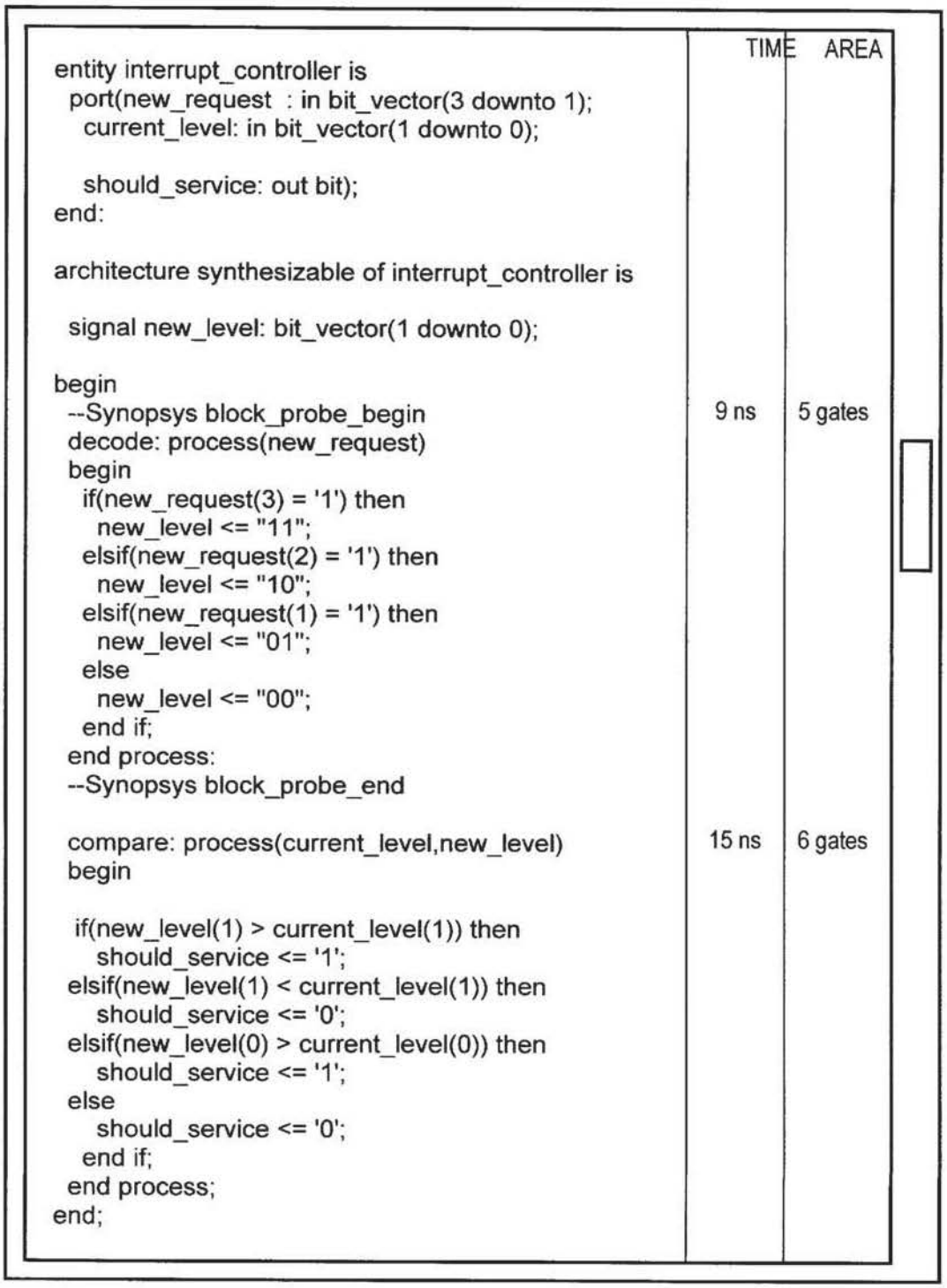
design : interrupt_controller	designer :	date : 4/5/94
technology : gtech	company :	sheet : 1 of 1

Figure 17



design : interrupt_controller	designer :	date : 4/5/94
technology : lsi_10k	company :	sheet : 1 of 1

Figure 18



```
entity interrupt_controller is
  port(new_request  : in bit_vector(3 downto 1);
    current_level: in bit_vector(1 downto 0);

    should_service: out bit);
end:

architecture synthesizable of interrupt_controller is

  signal new_level: bit_vector(1 downto 0);

begin
  --Synopsys block_probe_begin
  decode: process(new_request)
  begin
    if(new_request(3) = '1') then
      new_level <= "11";
    elsif(new_request(2) = '1') then
      new_level <= "10";
    elsif(new_request(1) = '1') then
      new_level <= "01";
    else
      new_level <= "00";
    end if;
  end process:
  --Synopsys block_probe_end

  compare: process(current_level,new_level)
  begin

   if(new_level(1) > current_level(1)) then
      should_service <= '1';
  elsif(new_level(1) < current_level(1)) then
      should_service <= '0';
  elsif(new_level(0) > current_level(0)) then
      should_service <= '1';
  else
      should_service <= '0';
    end if;
  end process;
end;
```

	TIME	AREA
--Synopsys block_probe_begin	9 ns	5 gates
compare: process(current_level,new_level)	15 ns	6 gates

Figure 19

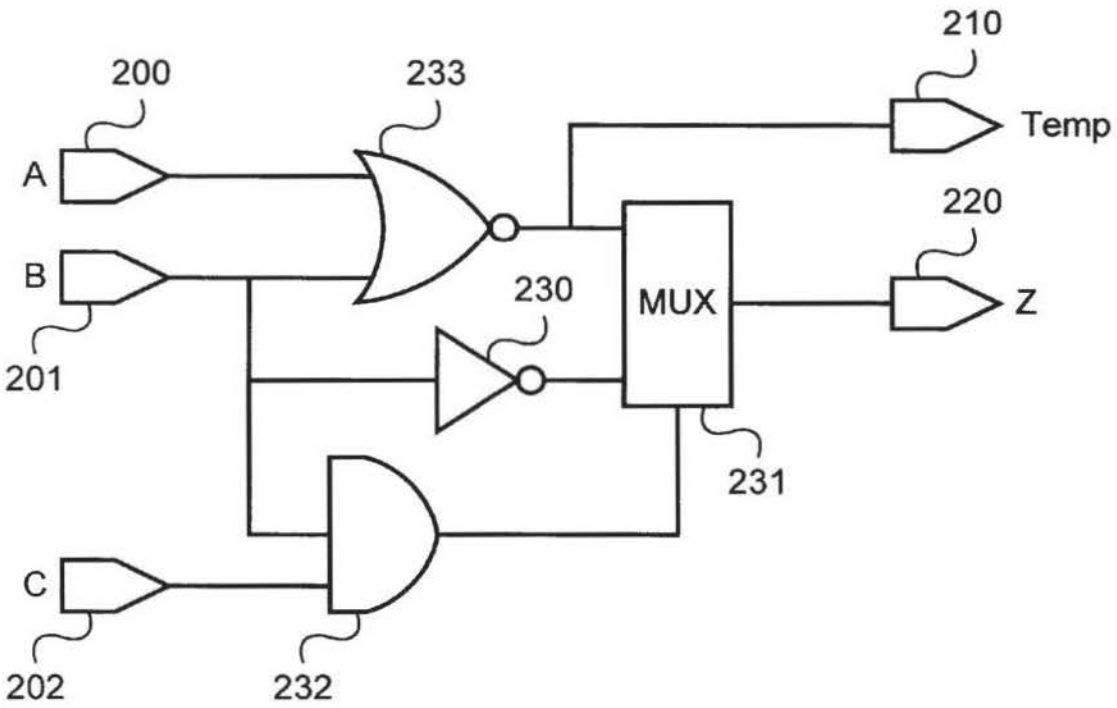


Figure 20

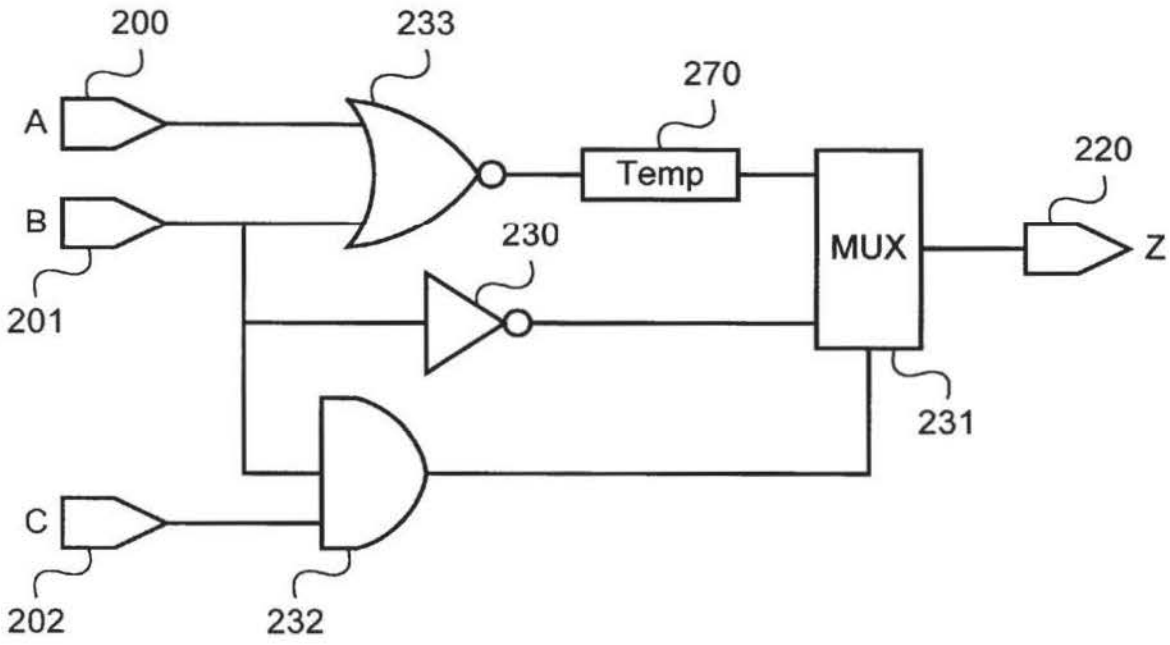


Figure 21

300

```
entity interrupt_controller is
  port(new_request  : in bit_vector(3 downto 1);
     current_level: in bit_vector(1 downto 0);

     should_service: out bit);
end;

architecture synthesizable of interrupt_controller is

  signal new_level: bit_vector(1 downto 0);

begin

  decode: process(new_request)
  begin
    if(new_request(3) = '1') then
      new_level <= "11";
    eisif(new_request(2) = '1') then
      new_level <= "10";
    eisif(new request(1) = '1') then
      new_level <= "01";
    else
      new_level <= "00";
    end if;
  end process;

  compare:process(current_level_new_level)
  begin

    if(new_level(1) > current_level(1)) then
      should_service <=  '1';
    eisif(new_level(1) < current_level(1)) then
      should_service <= '0';
    eisif(new_level(0) > current_level (0)) then
      should_service <= '1';
    else
      should_service <= '0';
    end if;

  end process;
end;
```

Figure 22

```
entity interrupt_controller is
 port(new_request  : in bit_vector(3 downto 1);
    current_level: in bit_vector(1 downto 0);

    should_service: out bit);
end;

architecture synthesizable of interrupt_controller is

 signal new_level: bit_vector(1 downto 0);

begin

 decode: process(new_request)
 begin
  if(new_request(3) = '1') then
    new_level <= "11";
  eisif(new_request(2) = '1') then
    new_level <= "10";
 eisif(new request(1) = '1') then
   new_level <= "01";
 else
   new_level <= "00";
 end if;
end process;

compare:process(current_level_new_level)
begin

  if(new_level(1) > current_level(1)) then
    should_service <=  '1';
  eisif(new_level(1) < current_level(1)) then
    should_service <= '0';
  eisif(new_level(0) > current_level (0)) then
    should_service <= '1';
  else
   should_service <= '0';
  end if;

 end process;
end;
```

300

Figure 23

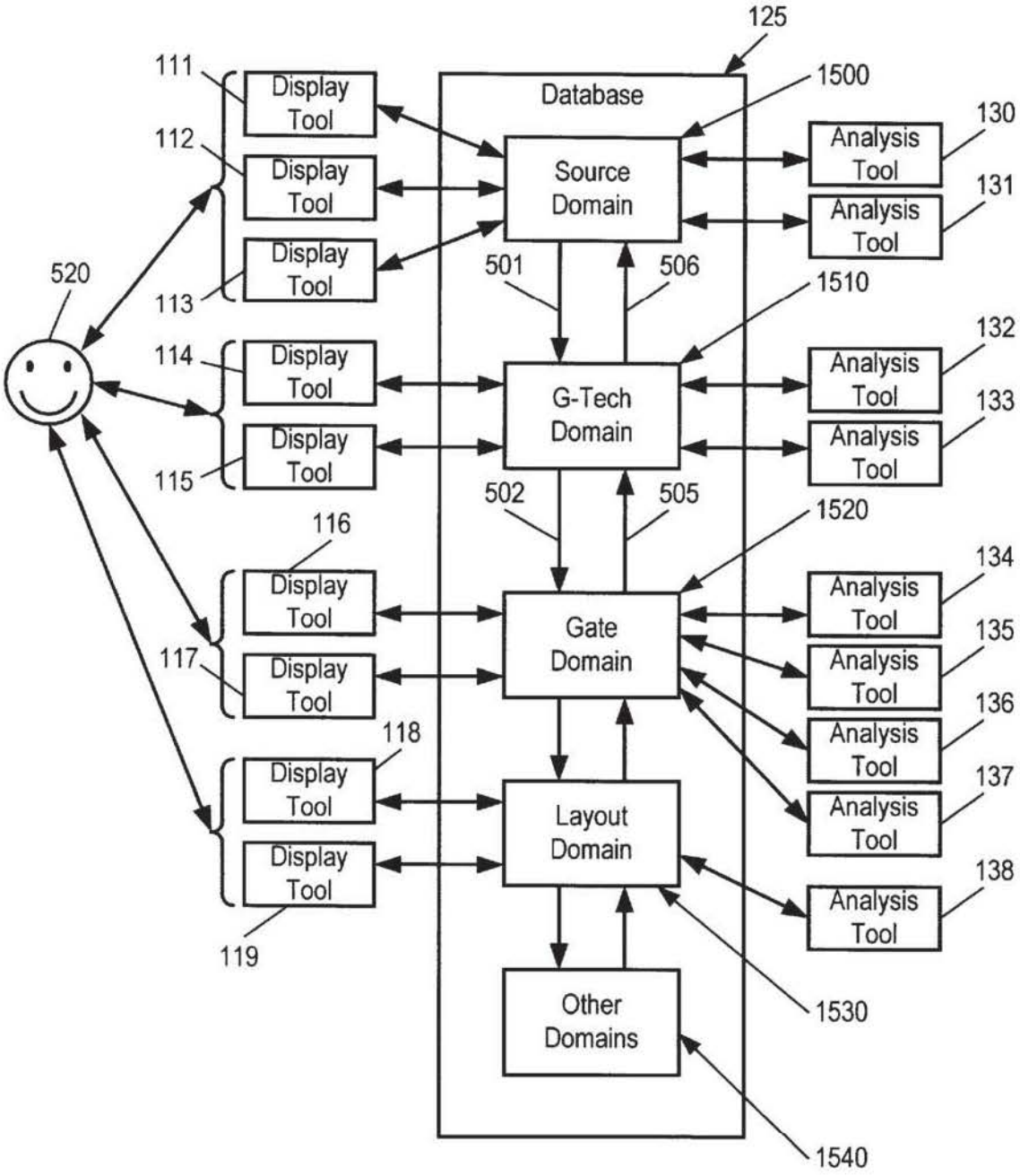

125
111
Display Tool
Database
1500
112
Display Tool
Source Domain
130
Analysis Tool
131
Analysis Tool
Display Tool
113
520
501
506
1510
114
Display Tool
G-Tech Domain
132
Analysis Tool
133
Analysis Tool
Display Tool
115
502
505
116
1520
Display Tool
Gate Domain
134
Analysis Tool
135
Analysis Tool
117
Display Tool
136
Analysis Tool
118
137
Analysis Tool
Display Tool
Layout Domain
Display Tool
138
Analysis Tool
119
Other Domains
1530
1540

Figure 24

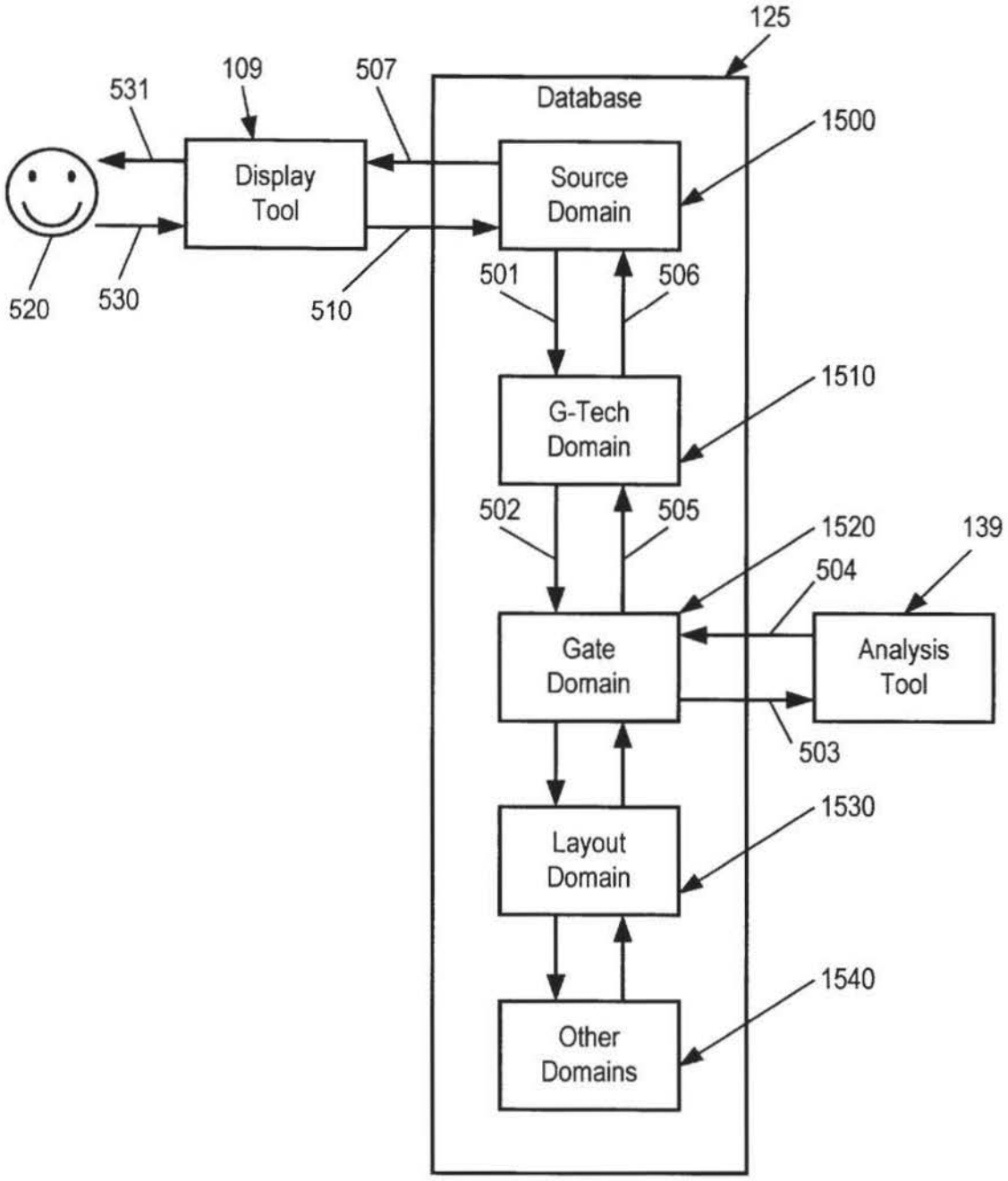

125
109
507
531
Database
1500
Display Tool
Source Domain
501
506
520
530
510
1510
G-Tech Domain
502
505
1520
139
504
Gate Domain
Analysis Tool
503
1530
Layout Domain
1540
Other Domains

Figure 25

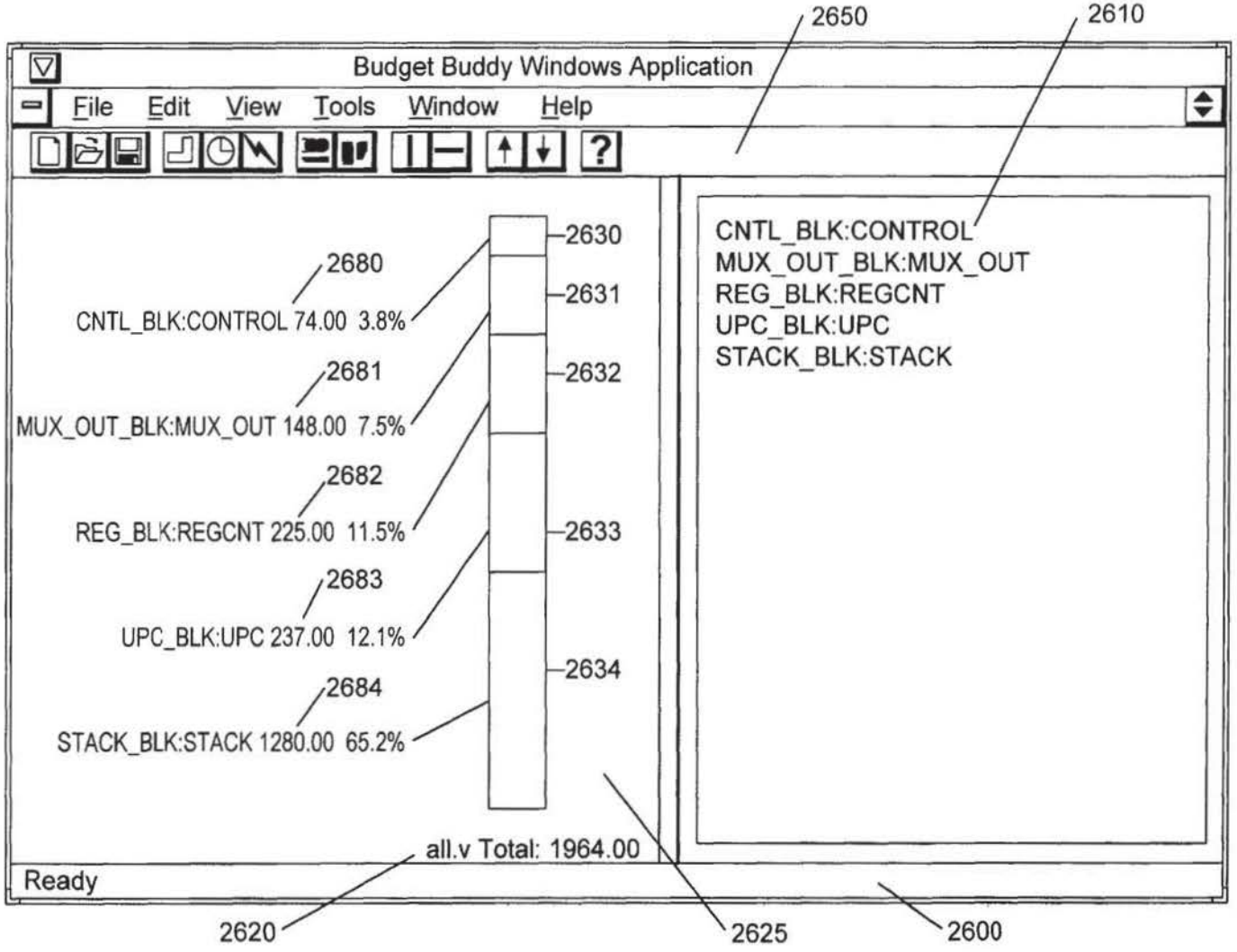

2650
2610
Budget Buddy Windows Application
File Edit View Tools Window Help
2630
2631
2632
2633
2634
2680
CNTL_BLK:CONTROL 74.00 3.8%
2681
MUX_OUT_BLK:MUX_OUT 148.00 7.5%
2682
REG_BLK:REGCNT 225.00 11.5%
2683
UPC_BLK:UPC 237.00 12.1%
2684
STACK_BLK:STACK 1280.00 65.2%
CNTL_BLK:CONTROL
MUX_OUT_BLK:MUX_OUT
REG_BLK:REGCNT
UPC_BLK:UPC
STACK_BLK:STACK
all.v Total: 1964.00
Ready
2620
2625
2600

Figure 26

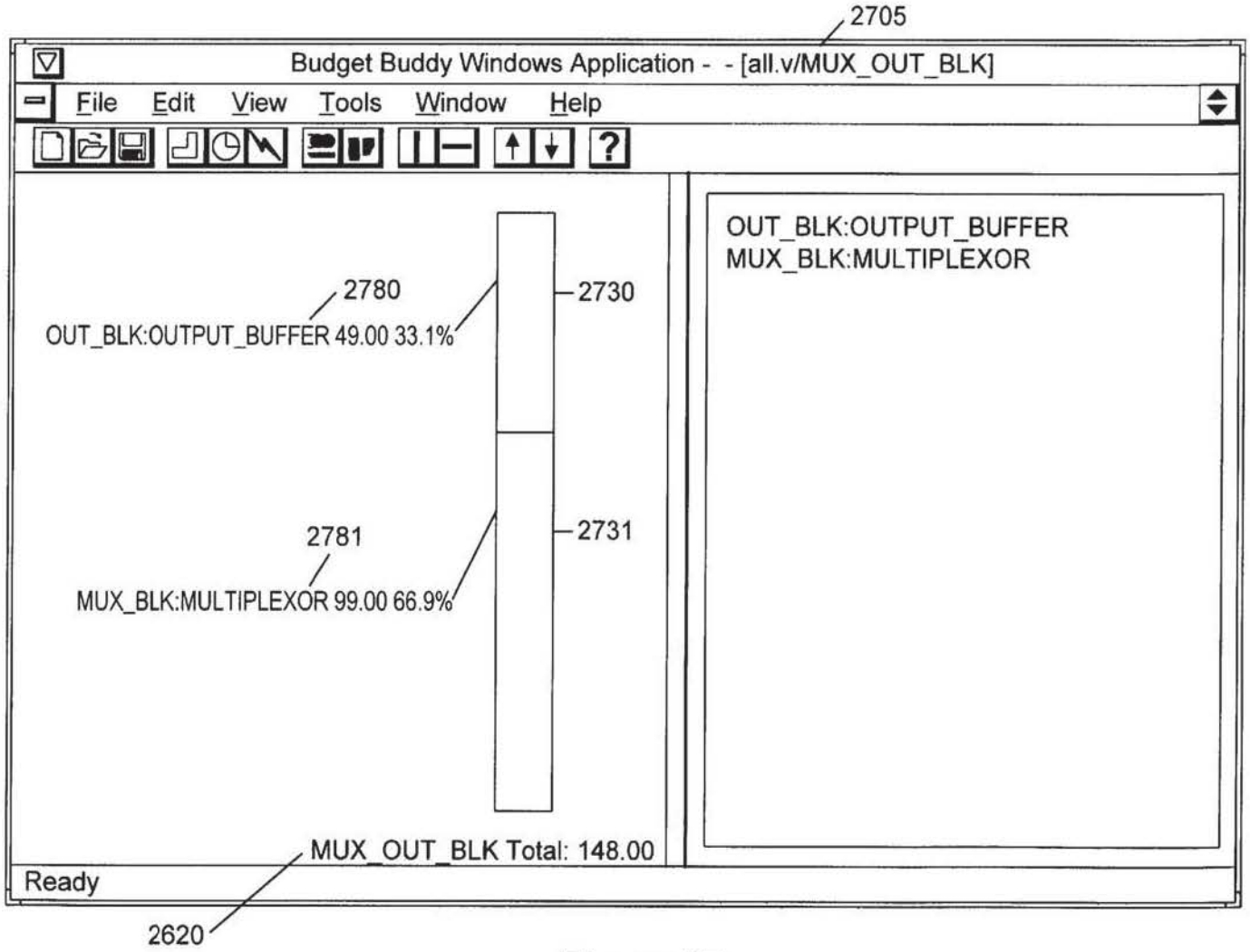

2705
Budget Buddy Windows Application - - [all.v/MUX_OUT_BLK]
File Edit View Tools Window Help
OUT_BLK:OUTPUT_BUFFER
MUX_BLK:MULTIPLEXOR
2780
OUT_BLK:OUTPUT_BUFFER 49.00 33.1%
2730
2781
MUX_BLK:MULTIPLEXOR 99.00 66.9%
2731
MUX_OUT_BLK Total: 148.00
Ready
2620

Figure 27

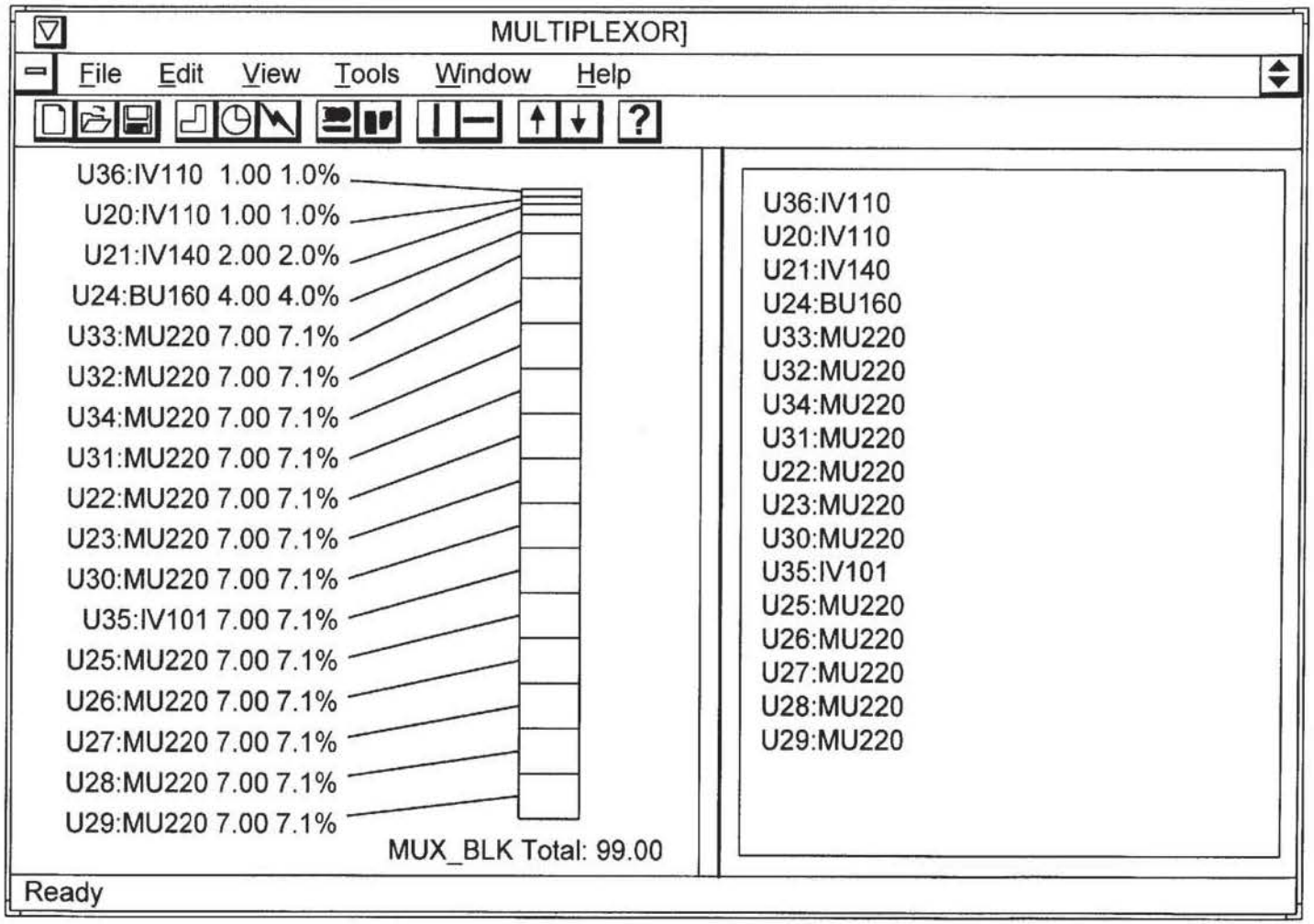

MULTIPLEXOR]
File Edit View Tools Window Help
U36:IV110 1.00 1.0%
U20:IV110 1.00 1.0%
U21:IV140 2.00 2.0%
U24:BU160 4.00 4.0%
U33:MU220 7.00 7.1%
U32:MU220 7.00 7.1%
U34:MU220 7.00 7.1%
U31:MU220 7.00 7.1%
U22:MU220 7.00 7.1%
U23:MU220 7.00 7.1%
U30:MU220 7.00 7.1%
U35:IV101 7.00 7.1%
U25:MU220 7.00 7.1%
U26:MU220 7.00 7.1%
U27:MU220 7.00 7.1%
U28:MU220 7.00 7.1%
U29:MU220 7.00 7.1%
MUX_BLK Total: 99.00
U36:IV110
U20:IV110
U21:IV140
U24:BU160
U33:MU220
U32:MU220
U34:MU220
U31:MU220
U22:MU220
U23:MU220
U30:MU220
U35:IV101
U25:MU220
U26:MU220
U27:MU220
U28:MU220
U29:MU220
Ready

Figure 28

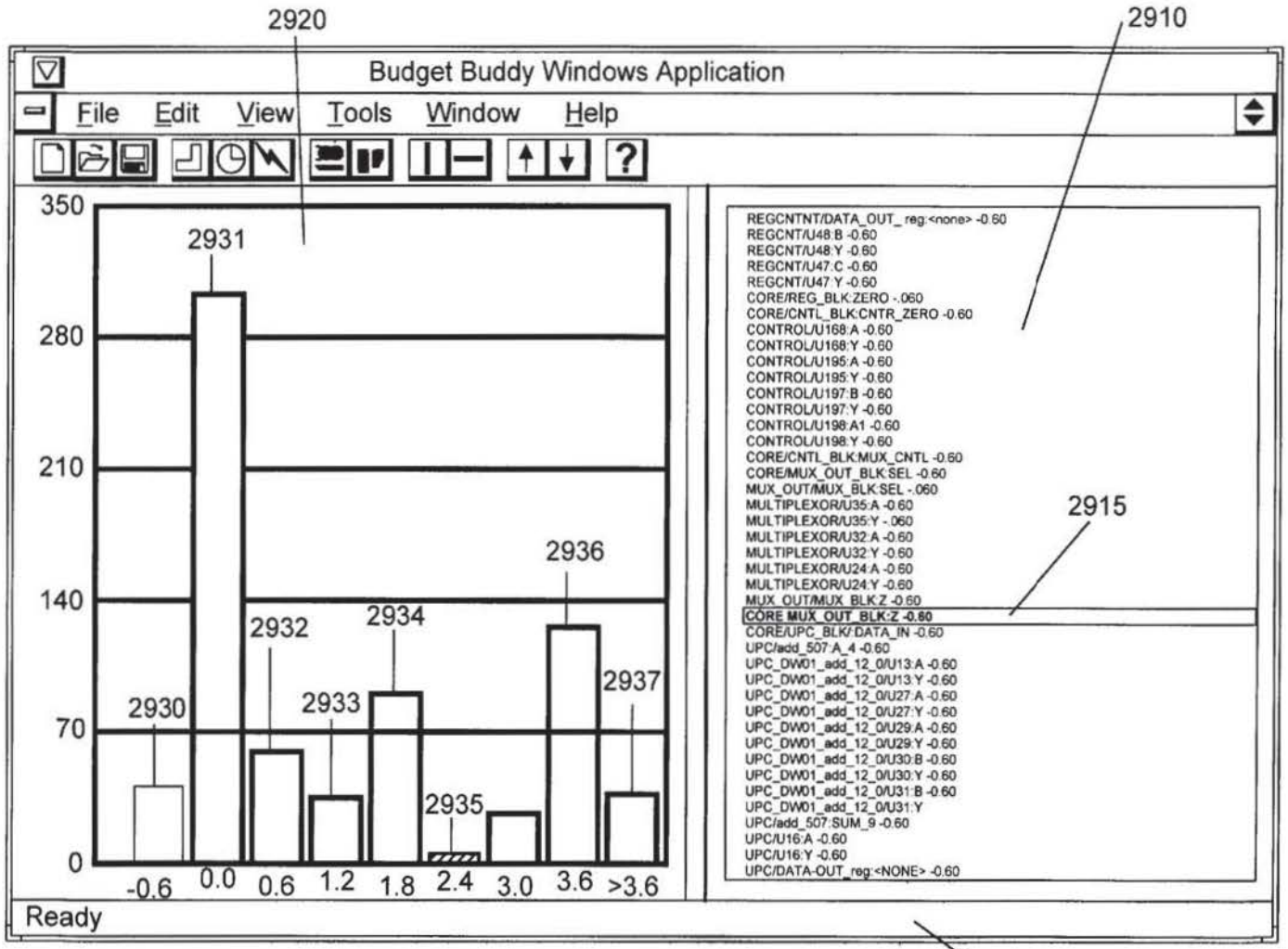

2920
2910
Budget Buddy Windows Application
File Edit View Tools Window Help
350
280
210
140
70
0
2931
2930
2932
2933
2934
2935
2936
2937
-0.6 0.0 0.6 1.2 1.8 2.4 3.0 3.6 >3.6
REGCNTNT/DATA_OUT_ reg:<none> -0.60
REGCNT/U48:B -0.60
REGCNT/U48:Y -0.60
REGCNT/U47:C -0.60
REGCNT/U47:Y -0.60
CORE/REG_BLK:ZERO -.060
CORE/CNTL_BLK:CNTR_ZERO -0.60
CONTROL/U168:A -0.60
CONTROL/U168:Y -0.60
CONTROL/U195:A -0.60
CONTROL/U195:Y -0.60
CONTROL/U197:B -0.60
CONTROL/U197:Y -0.60
CONTROL/U198:A1 -0.60
CONTROL/U198:Y -0.60
CORE/CNTL_BLK:MUX_CNTL -0.60
CORE/MUX_OUT_BLK:SEL -0.60
MUX_OUT/MUX_BLK:SEL -.060
MULTIPLEXOR/U35:A -0.60
MULTIPLEXOR/U35:Y -.060
MULTIPLEXOR/U32:A -0.60
MULTIPLEXOR/U32:Y -0.60
MULTIPLEXOR/U24:A -0.60
MULTIPLEXOR/U24:Y -0.60
MUX_OUT/MUX_BLK:Z -0.60
CORE MUX_OUT_BLK:Z -0.60
CORE/UPC_BLK/:DATA_IN -0.60
UPC/add_507:A_4 -0.60
UPC_DW01_add_12_0/U13:A -0.60
UPC_DW01_add_12_0/U13:Y -0.60
UPC_DW01_add_12_0/U27:A -0.60
UPC_DW01_add_12_0/U27:Y -0.60
UPC_DW01_add_12_0/U29:A -0.60
UPC_DW01_add_12_0/U29:Y -0.60
UPC_DW01_add_12_0/U30:B -0.60
UPC_DW01_add_12_0/U30:Y -0.60
UPC_DW01_add_12_0/U31:B -0.60
UPC_DW01_add_12_0/U31:Y
UPC/add_507:SUM_9 -0.60
UPC/U16:A -0.60
UPC/U16:Y -0.60
UPC/DATA-OUT_reg:<NONE> -0.60
2915
Ready
2900

Figure 29

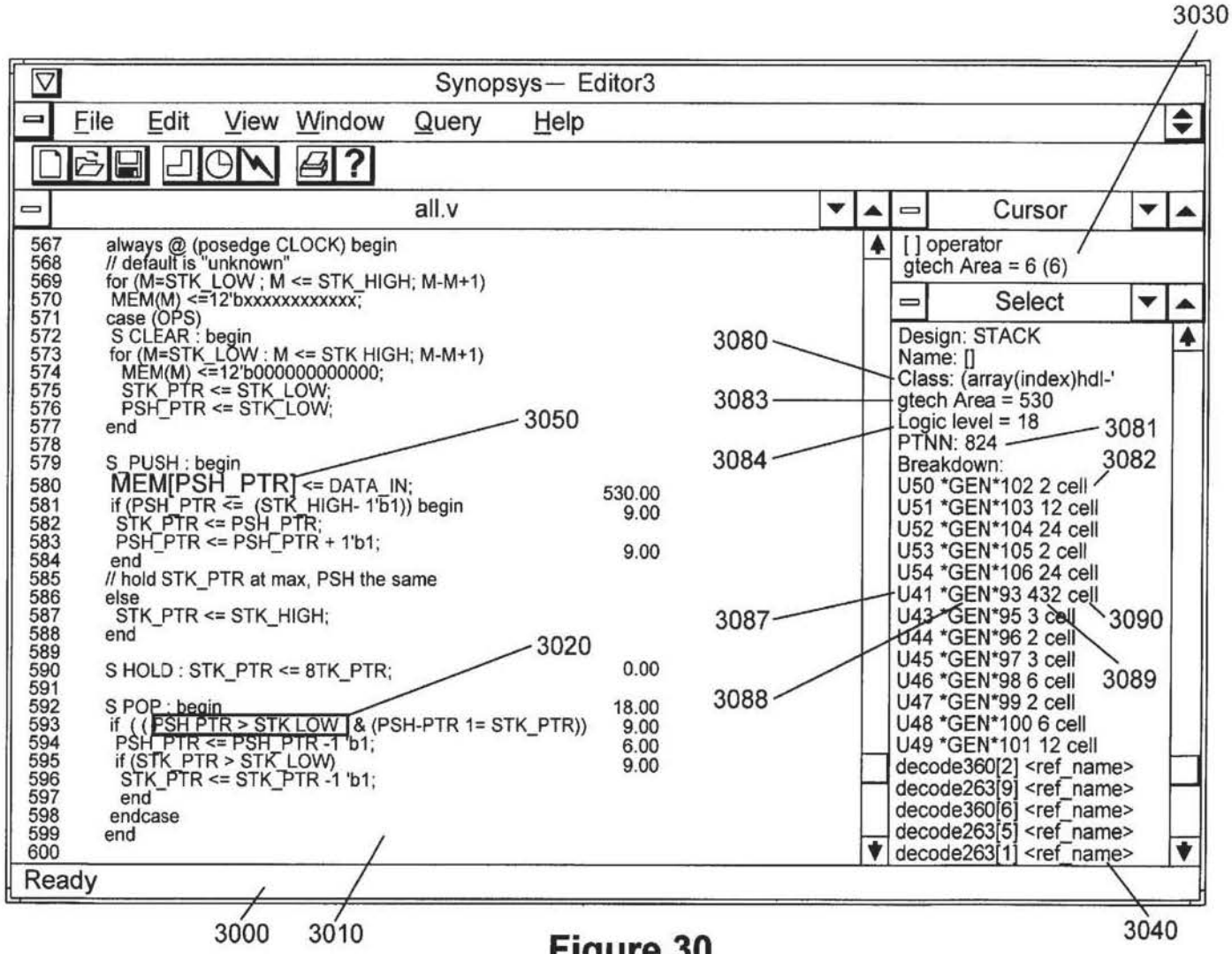

Synopsys— Editor3
File Edit View Window Query Help
all.v
Cursor
[ ] operator
gtech Area = 6 (6)
Select
Design: STACK
Name: []
Class: (array(index)hdl-'
gtech Area = 530
Logic level = 18
PTNN: 824
Breakdown:
U50 *GEN*102 2 cell
U51 *GEN*103 12 cell
U52 *GEN*104 24 cell
U53 *GEN*105 2 cell
U54 *GEN*106 24 cell
U41 *GEN*93 432 cell
U43 *GEN*95 3 cell
U44 *GEN*96 2 cell
U45 *GEN*97 3 cell
U46 *GEN*98 6 cell
U47 *GEN*99 2 cell
U48 *GEN*100 6 cell
U49 *GEN*101 12 cell
decode360[2] <ref_name>
decode263[9] <ref_name>
decode360[6] <ref_name>
decode263[5] <ref_name>
decode263[1] <ref_name>
567 always @ (posedge CLOCK) begin
568 // default is "unknown"
569 for (M=STK_LOW ; M <= STK_HIGH; M-M+1)
570 MEM(M) <=12'bxxxxxxxxxxxx;
571 case (OPS)
572 S CLEAR : begin
573 for (M=STK_LOW : M <= STK HIGH; M-M+1)
574 MEM(M) <=12'b000000000000;
575 STK_PTR <= STK_LOW;
576 PSH_PTR <= STK_LOW;
577 end
578
579 S_PUSH : begin
580 MEM[PSH_PTR] <= DATA_IN;
581 if (PSH_PTR <= (STK_HIGH- 1'b1)) begin
582 STK_PTR <= PSH_PTR;
583 PSH_PTR <= PSH_PTR + 1'b1;
584 end
585 // hold STK_PTR at max, PSH the same
586 else
587 STK_PTR <= STK_HIGH;
588 end
589
590 S HOLD : STK_PTR <= 8TK_PTR;
591
592 S POP : begin
593 if ((PSH PTR > STK LOW) & (PSH-PTR 1= STK_PTR))
594 PSH_PTR <= PSH_PTR -1 'b1;
595 if (STK_PTR > STK_LOW)
596 STK_PTR <= STK_PTR -1 'b1;
597 end
598 endcase
599 end
600
530.00
9.00
9.00
0.00
18.00
9.00
6.00
9.00
Ready
3030
3080
3083
3084
3050
3081
3082
3087
3090
3089
3088
3020
3000
3010
3040

Figure 30

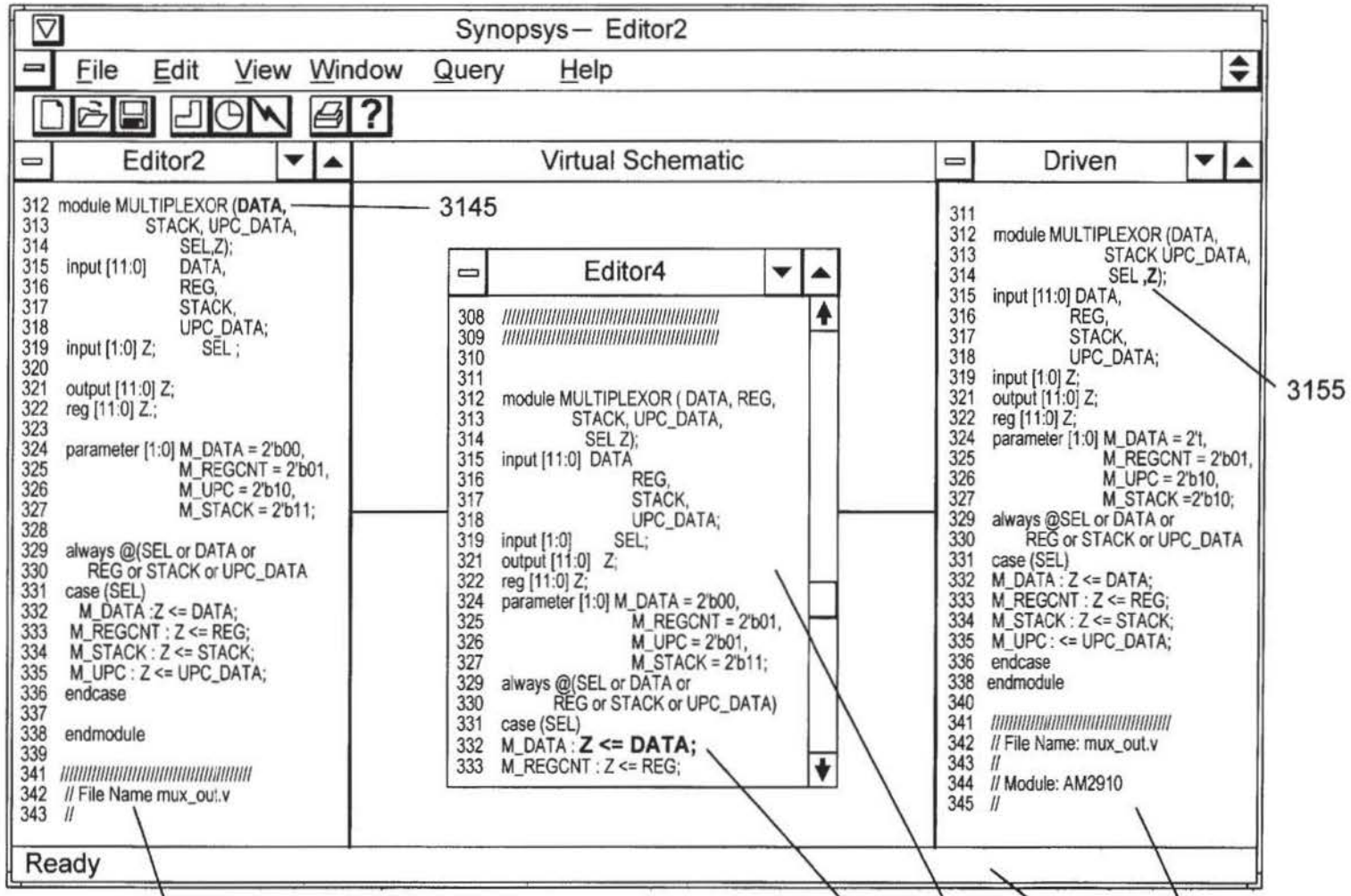

Synopsys— Editor2
File Edit View Window Query Help
Editor2
Virtual Schematic
Driven
3145
3155
Editor4
Ready
3110
3150
3120
3100
3130

Figure 31

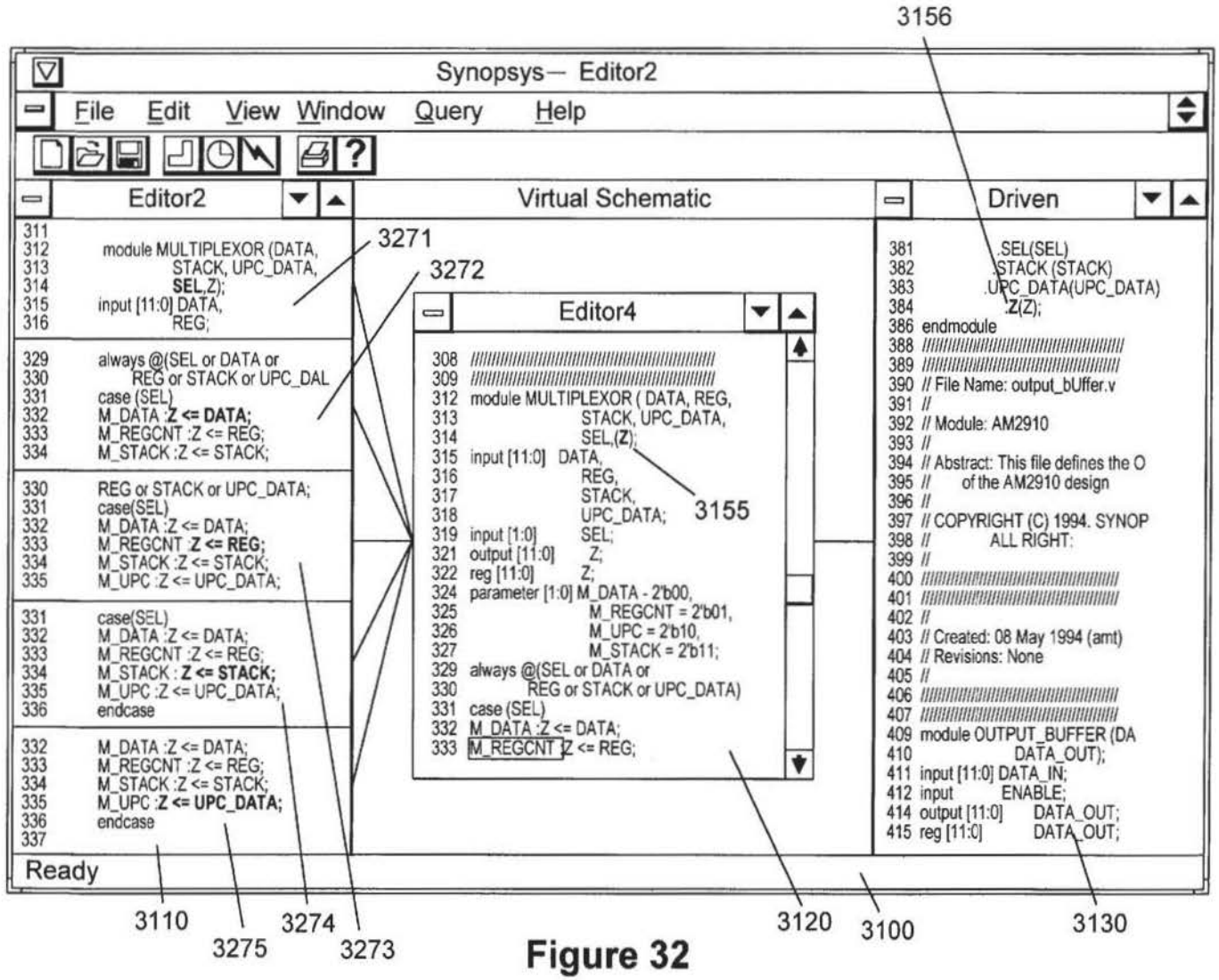

3156
Synopsys— Editor2
File Edit View Window Query Help
Editor2
Virtual Schematic
Driven
3271
3272
Editor4
3155
Ready
3110
3275
3274
3273
3120
3100
3130

Figure 32

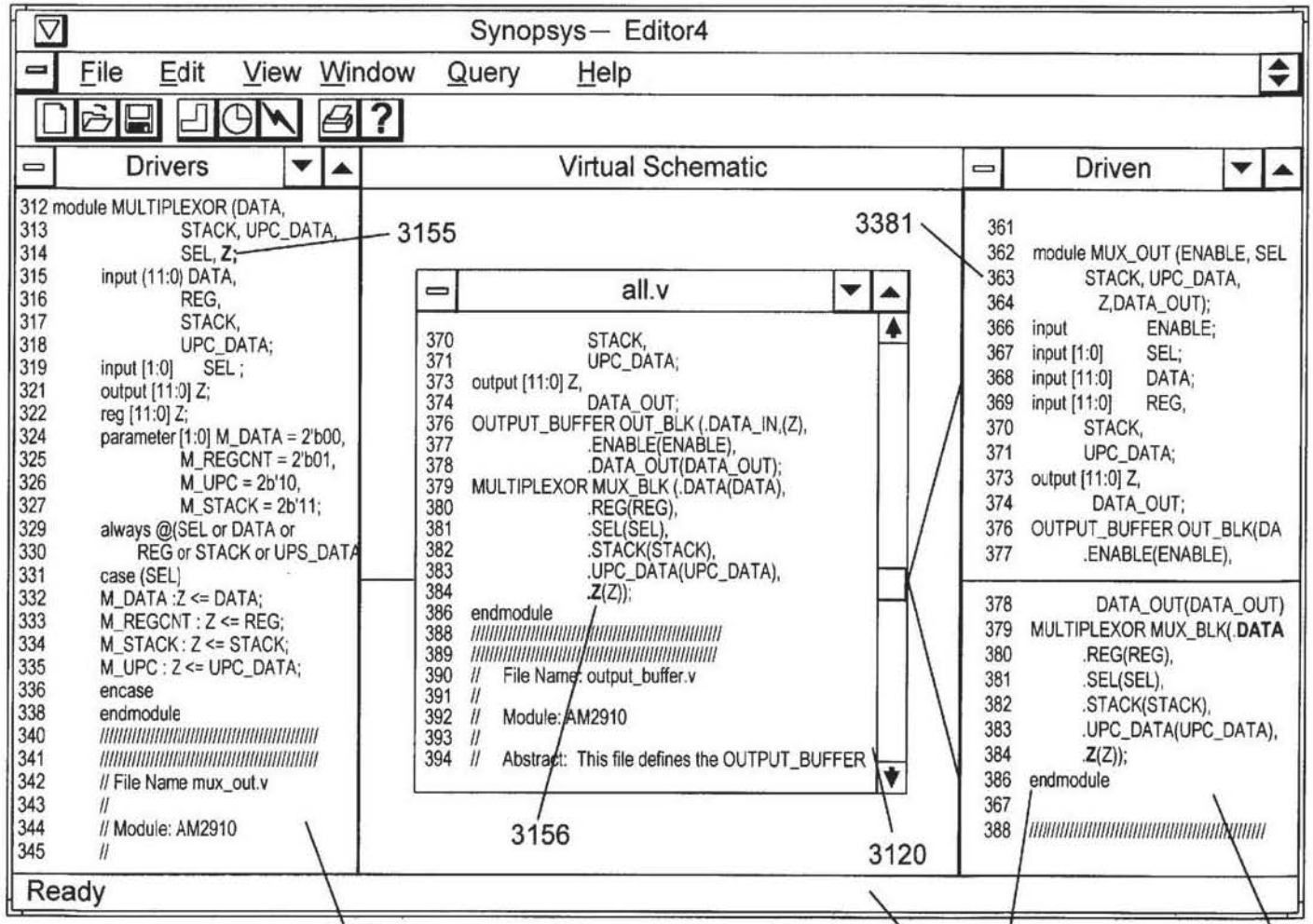

Synopsys— Editor4
File Edit View Window Query Help
Drivers
Virtual Schematic
Driven
312 module MULTIPLEXOR (DATA,
313 STACK, UPC_DATA,
314 SEL, Z);
315 input (11:0) DATA,
316 REG,
317 STACK,
318 UPC_DATA;
319 input [1:0] SEL ;
321 output [11:0] Z;
322 reg [11:0] Z;
324 parameter [1:0] M_DATA = 2'b00,
325 M_REGCNT = 2'b01,
326 M_UPC = 2b'10,
327 M_STACK = 2b'11;
329 always @(SEL or DATA or
330 REG or STACK or UPS_DATA
331 case (SEL)
332 M_DATA :Z <= DATA;
333 M_REGCNT : Z <= REG;
334 M_STACK : Z <= STACK;
335 M_UPC : Z <= UPC_DATA;
336 encase
338 endmodule
340 //
341 //
342 // File Name mux_out.v
343 //
344 // Module: AM2910
345 //
3155
3381
all.v
370 STACK,
371 UPC_DATA;
373 output [11:0] Z,
374 DATA_OUT;
376 OUTPUT_BUFFER OUT_BLK (.DATA_IN,(Z),
377 .ENABLE(ENABLE),
378 .DATA_OUT(DATA_OUT);
379 MULTIPLEXOR MUX_BLK (.DATA(DATA),
380 .REG(REG),
381 .SEL(SEL),
382 .STACK(STACK),
383 .UPC_DATA(UPC_DATA),
384 .Z(Z));
386 endmodule
388 //
389 //
390 // File Name: output_buffer.v
391 //
392 // Module: AM2910
393 //
394 // Abstract: This file defines the OUTPUT_BUFFER
3156
3120
361
362 module MUX_OUT (ENABLE, SEL
363 STACK, UPC_DATA,
364 Z,DATA_OUT);
366 input ENABLE;
367 input [1:0] SEL;
368 input [11:0] DATA;
369 input [11:0] REG,
370 STACK,
371 UPC_DATA;
373 output [11:0] Z,
374 DATA_OUT;
376 OUTPUT_BUFFER OUT_BLK(DA
377 .ENABLE(ENABLE),
378 DATA_OUT(DATA_OUT)
379 MULTIPLEXOR MUX_BLK(.DATA
380 .REG(REG),
381 .SEL(SEL),
382 .STACK(STACK),
383 .UPC_DATA(UPC_DATA),
384 .Z(Z));
386 endmodule
367
388 //
Ready
3110
3100
3382
3130

Figure 33

ARCHITECTURE AND METHODS FOR A HARDWARE DESCRIPTION LANGUAGE SOURCE LEVEL DEBUGGING SYSTEM

RELATED APPLICATION

This application is a continuation-in-part of U.S. application Ser. No. 08/226,147 by Gregory et al, filed on Apr. 12, 1994, now abandoned.

BACKGROUND

1. Field of the Invention

This invention relates to the field of computer aided design for digital circuits, and particularly to debugging digital circuits constructed using logic or behavioral synthesis. This invention also relates to displaying the results of circuit analysis determined in one domain, such as an annotated netlist of gates, in the context of the circuit structure in another domain, such as the hardware description language (HDL) source text.

2. Statement of the Related Art

A digital circuit designer needs to ensure that the circuit performs the correct function subject to many design constraints. For example, the circuit should perform the correct computation in the proper amount of time. The area that the circuit occupies on a semiconductor die should remain within certain bounds. The power that the circuit consumes while operating should also remain within specified bounds. To be economically manufacturable, the circuit should be testable. An economically useful circuit should not take too long to design, manufacture, test or use.

The digital circuit design process involves translating the designer's sometimes ambiguous thoughts about the function and constraints into the tooling necessary to produce a working circuit. For example, producing a full-custom semiconductor chip requires producing masks that define the deposition of chemicals into a substrate as well as producing test patterns that exercise the final product. As another example of tooling, producing a field programmable gate array requires generating the bit pattern to be downloaded into the chip to specify the configuration of the architecture. Computer Aided Design (CAD) tools facilitate the iterative translation of the designer's developing thoughts into the tooling required to produce a working circuit that satisfies the design constraints. The process of iteratively adjusting a design to meet its requirements is called debugging.

The historical model of the digital design process using conventional CAD tools for a semi-conductor chip is as follows. The designer first conceives of a particular function to implement, as well as constraints such as timing or area that the implementation must meet. Next, historically, the designer mentally transforms the desired function into a high level circuit consisting of components such as gates, adders, registers and RAMS. The designer then captures that insight by drawing a schematic of a circuit that implements that function with a schematic capture tool. The schematic shows how more primitive functional elements, such as gates or transistors, connect together to form more sophisticated functions such as arithmetic logic units. In addition, modern schematic capture tools allow the designer to divide the design hierarchically into interconnected pieces, and then allow the user to specify the details of each of the pieces separately. For example, Design Architect by Mentor Graphics of Wilsonville, Oregon provides these schematic capture functions.

Conventional CAD tools, such as those indicated above, can then take the connections in the schematic and other

information to evaluate the circuit and to specify the tooling necessary to construct the circuit. Such tools evaluate the circuit in many ways. For example, commercial CAD tools often have a simulator that predicts the response of the circuit to designer specified input patterns. QuickSim II by Mentor Graphics of Wilsonville, Oreg. is a commonly used simulator. Another common CAD tool is a path delay analyzer that identifies the longest timing path in a circuit design. DesignTime by Synopsys, Inc. of Mountain View, Calif. is a tool that provides path delay analysis.

Conventional CAD tools also have the ability to generate the geometric layout of the circuit with layout tools. Cell3 Ensemble by Cadence of San Jose, Calif. is an example of this type of tool. Layout tools are required to produce masks to make a semi-conductor chip.

Conventional CAD tools also have the ability to check that the circuit meets the design rules, and to identify the location of any errors to the designer. Design rules help ensure that the specified circuit will operate once manufactured.

Conventional CAD tools also are used to determine how testable a circuit is, and to generate test patterns automatically. Showing the designer the parts of the circuit that are not testable allows the designer to make modifications that will increase the probability of making a successful chip or circuit. Generating test patterns automatically allows for more thorough testing of the circuit immediately after manufacturing.

As described above, the concept of debugging a circuit design historically refers to the process by which a circuit designer specified a particular implementation with a schematic capture tool, and then used various circuit evaluation tools to verify that the implementation did what the circuit designer wanted. For example, the designer would use a simulator to determine if the circuit produced appropriate outputs from specified inputs. The designer could use the path delay analyzer to determine whether the current design was fast enough to meet the requirements. The layout tools could inform the designer whether the design would fit in the available space.

When a particular design did not meet the designer's expectations or requirements, the designer then modified the design. For example, if the circuit was too slow, the designer identified the offending circuitry and revamped it to increase performance. If the circuit was too large, then the designer revised the circuit to use fewer or smaller components. If the circuit did not behave as required, the designer changed the components and the interconnections to produce the correct function. Because the conventional CAD tools began the analysis with the captured circuit, the timing or area problems could be readily identified to the designer. Because the designer specified the structure of the circuit, the designer could then thoughtfully make adjustments. However, because historically the designer mentally performed the transformation from desired function to circuit, the CAD tools were limited in their ability to identify functional problems to the designer.

Logic synthesis developed to provide the designer with an automatic mechanism to translate a hardware description language (HDL) description of a desired function to a structural description of a circuit that performed the desired function. Logic synthesis begins with the designer describing the desired function using VHDL, Verilog, or any other logic synthesis source language, to specify the behavior. A translator then converts that description into gates and other circuit structures that directly correspond statement by state-

ment with the designer's description. Theoretically, conventional CAD tools could then evaluate the resulting circuit and develop the appropriate tooling.

However, the circuit created by a statement-by-statement translation is generally large and slow. In logic synthesis, translation is followed by logic optimization. Optimization replaces the directly translated structure with a functionally equivalent, yet improved structure.

Unfortunately, the circuit transformations performed by the logic optimizer usually modifies the structure of the circuit. This results in a circuit that is unrecognizable by the designer. The fact that the designer generally can not recognize the original function performed by the transformed circuit makes debugging synthesized logic difficult. Conventional evaluation tools can determine the timing or area problems in the transformed circuit, but the designer can not relate those problems easily to the HDL source specification. Theoretically, the designer could manually determine what part of the HDL specification caused the problem. With that insight, the designer could make the desired changes at the HDL specification, and resynthesize the entire circuit. If the designer's problem occurred in a part of the circuit that passed through the optimizer with few changes, manual backtracking might work. However, the optimization process generally makes many changes, making it either difficult or impossible to backtrack because many points in the original circuit do not exist in the final circuit.

Alternatively, the designer could simply modify the final circuit directly. This would not allow the designer to resynthesize the design from the HDL specification because the designer's circuit changes would be overwritten by subsequent translation and optimization steps. This would destroy the value gained by using the synthesis approach to design.

There are some existing tools and techniques for determining where and how to modify a HDL source specification. One tool allows the designer to examine a gate in the optimized circuit schematic, and inquire where that gate came from in the HDL source, provided that it directly existed in the source. If the tool could not tell where a gate came from, it would say so. An example of a gate tracing tool is Design Analyzer's source-to-gates function, produced by Synopsys, Inc. of Mountain View, Calif. However, many gates are removed and others are added during the translation and optimization process. The gate tracing tool has not proven to be very useful in helping designers debug circuits because many of the components in a optimized circuit can not be traced back to the HDL source specification.

Another method of debugging a synthesized circuit is to partition the design into hierarchical components, and synthesize and optimize the smaller pieces. Because the synthesis and optimization tools generally do not traverse primary inputs and outputs, such a partitioning can reduce the size of the backtracking problem. However, it has the disadvantage that the designer may have to rewrite functionally correct, but nonetheless problematic, synthesis source code to isolate the troublesome parts of the circuit. In addition, this approach will greatly limit the optimizer's ability to reduce the area and increase the speed of the resulting circuits because the optimizer will be constrained by the designer's partition.

In addition, a designer can be mislead by the results obtained by debugging by partitioning. The designer's bug in the circuit might be that it is too slow or too big. Partitioning the synthesis source to locate the cause will result in a different circuit than the unpartitioned source. Therefore, the problem that the designer is chasing could vanish or be made significantly worse by the debugging process itself.

Conventionally, using a synthesizer to translate a specification into a circut can also raise substantial computational problems to incorporate minor changes into a design late in the design process. For example, a designer could have the design fairly close to completion when the designer discovers the need to make a small functional change, such as inverting a particular signal. Intuitively, one would expect that such a small change would require only a small change in the circuit all the way to the layout level. However, it is quite possible that, with conventional synthesis and optimization tools, a small change could require substantial changes in the circuit and the layout. Currently, a designer can limit this kind of problem by partitioning the design into smaller pieces and thus limiting the effect to the directly implicated pieces. However, as described previously, inappropriate or unduly narrow partitioning can limit the ability of the optimization tools to construct a good circuit.

3. A Conventional Design and Debugging Process Overview

FIG. **1** shows an overview of the conventional process for designing and debugging circuits specified with a Hardware Description Language (HDL). The process begins with the designer writing HDL source code **100**. A typical language used for specifying circuits is VHDL which is described in the IEEE Standard VHDL Language Reference Manual available from the Institute of Electrical and Electronic Engineers in Piscataway, N.J., which is hereby incorporated by reference. VHDL stands for Very high speed integrated circuit Hardware Description Language. Another language used for specifying circuits is Verilog that is described in Hardware Modeling with Verilog HDL by Eliezer Sternheim, Rajvir Singh, and Yatin Trivedi, published by Automata Publishing Company, Palo Alto, Calif., 1990, which is hereby incorporated by reference. Verilog is also described in the Verilog Hardware Description Language Reference Manual (LRM), version 1.0, November 1991, which is published by Open Verilog International, and is hereby incorporated by reference. The examples used in this document are in VHDL, but the principles readily apply to other circuit specification languages.

After writing a HDL description of a desired function, the designer then simulates the function **101** embedded in the description with a HDL simulator. An example of a functional simulator is VHDL System Simulator that is available from Synopsys, Inc. of Mountain View, Calif. The functional simulator allows the designer to determine whether the circuit produces correct values in response to inputs without regard to timing, area or power constraints. A functional simulator can perform function-only simulation relatively quickly, thus enabling the designer to determine that the circuit will produce the desired output.

If there is a problem with the function, the designer can fix function problems **102** by examining the simulation output and going back to writing HDL code **100**. Functional simulation executes the source specification directly without generating or optimizing circuits. Therefore, problems identified during functional simulation can readily be linked to their cause in the HDL source.

If the designer believes that the described circuit provides the correct function, then designer then specifies constraints for the synthesis process **103**, e.g. maximum clocking periods, total circuit area, and maximum power. This part of the process is described in Design Compiler Family Reference Manual, Version 3.1a, which is available from Synopsys, Inc. of Mountain View, Calif. and is hereby incorporated by reference. Examples of Computer Aided

Design software that use constraint specification are Synergy by Cadence, and Autologic by Mentor Graphics, and Design Compiler by Synopsys.

After developing constraints, the designer then proceeds to synthesize **104** a circuit from the HDL description produced in the writing HDL **100** step. This step involves translating the description into an initial circuit that correspond directly with the statements in the source HDL. An example of software that performs this function is described in the VHDL Compiler Reference Manual, Version 3.1a, which is available from Synopsys, and is hereby incorporated by reference. After translation, the initial circuit is then optimized into a final circuit that meets the performance constraints established in step **103**. Prior to optimization, it is a straight-forward task to identify which circuit element of the initial circuit corresponds to what part of the HDL source code. Conventionally, because of the extensive manipulations performed during the optimization process, such identification after optimization becomes almost impossible except at registers and module interface boundaries.

FIG. **2** shows the intermediate data structures involved in the synthesis process **104**. The synthesis process begins with HDL source **900**. The translator creates a data structure called a parse tree **901** that represents the organizational structure of the HDL. The translator then turns the parse tree into an initial circuit **902**. Russ B. Segal's Master's Thesis, "BDSYN: Logic Design Translator" at the University of California at Berkeley, Memo#UCB/ERL M87/33, describes a such a translator, and is hereby incorporated by reference. United States patent application Ser. No. 07/632,439, filed on Dec. 21, 1990, entitled "Method and Apparatus for Synthesizing HDL Descriptions with Conditional Assignments" by Gregory et al, and commonly assigned to Synopsys, Inc. also describes such a translator, and is hereby incorporated by reference. An example of a tool that does this is version 3.1a of the HDL compiler available from Synopsys, Inc.

An optimizer is used to produce the final circuit **903** from the initial circuit **902**. The optimization process is explained in "Logic Synthesis Through Local Transformations" by J. Darringer, W. Joyner, L. Berman, and L. Trevillyan in the IBM Journal of Research and Development, volume 25, number 4, July 1981, pages 272–280, which is hereby incorporated by reference. It is also explained in "LSS: A System for Production Logic Synthesis" by J. Darringer, D. Brand, J. Gerbi, W. Joyner, and L. Trevillyan in the IBM Journal of Research and Development, volume 28, number 5, September 1984, pages 537–545, which is hereby incorporated by reference. It is also explained in "MIS: A Multiple-Level Logic Optimization System" by R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang in the IEEE Transactions on Computer Aided Design, Volume 6, number 6, November 1987, pages 1062–1081, which is hereby incorporated by reference. It is also explained in the Ph.D. dissertation "Logic Synthesis for VLSI Design" by R. Rudell at the University of California at Berkeley in 1989, which is hereby incorporated by reference. The optimization process is also described in the Design Compiler Family Reference Manual, version 3.1a which is available from Synopsys, and is hereby incorporated by reference. An example of software that performs this function is the Design Compiler available from Synopsys, Inc. Other examples of software that performs optimization include Synergy from Cadence, Inc., and AutoLogic by Mentor Graphics.

One approach to optimization is to group one or more initial circuit elements together, and replace those elements with a functionally equivalent collection of elements that has better characteristics than the collection of elements replaced. This results in an intermediate circuit that is functionally equivalent to the original. This intermediate circuit can then some or all of its elements grouped for another replacement. This process is repeated until the optimizer meets the constraints imposed in step **103** of FIG. **1**, or is unable to make any further improvement.

Before beginning the optimization process, the components of the initial circuit can be divided into two groups: those components that must be preserved through the optimization process, and those that can be replaced with functional equivalents. For example, a logic optimizer may replace a block of boolean logic with another block so long as function is maintained. Generally, replaceable components can also be eliminated. Examples of components that are generally preserved through the optimization process are primary inputs, primary outputs and registers.

After developing a circuit, the designer can then analyze the circuit **105** using conventional analysis tools, as shown in FIG. **1**. For example, the designer could estimate the area that the circuit consumes or what the longest delay path is in the circuit. This analysis can identify problems to the designer. The analysis report **904** is often a text document, as shown in FIG. **2**.

After identifying timing, area, testing or power problems with the analysis tools, the designer then devises a means to adjust the circuit to fix these problems **108**. Ideally, the designer would go back to the HDL where the function is specified and make adjustments there. However, because it is currently hard to identify the specific places in the source HDL that led to the problem, modifying the appropriate part of the HDL is currently not an effective debugging technique. The designer can usually identify which hierarchical module contains some, of the problem, and the designer could then manually rewrite that module to create more primary inputs and outputs to examine. This is very time consuming and is generally done as a last resort. Alternatively, the designer could adjust the constraints **103** and synthesizes the circuit **104** again to see if the problem is alleviated.

After debugging the circuit, the designer then releases the design for fabrication **106**.

4. System Performance

In addition to the debugging problems presented by the transformations made by the logic synthesis process, there are also difficulties associated with efficiently and economically constructing CAD systems that compute and display analysis results. Conceptually, after specifying a design, debugging a circuit involves having the designer repeatedly (1) determine a particular circuit characteristic or property that the designer wants to know about, (2) identify a kind of analysis that will provide information about that characteristic, (3) instruct the CAD system to perform that analysis, (4) display the results of that analysis, and (5) gain insight into the desired characteristic from the display. The designer is interested in completing these steps as quickly as possible. Circuit CAD tools have historically facilitated this goal by making the instruction and display steps computationally efficient. To improve response times, circuit CAD tools have often tightly couple the software that performed the analysis to the software that performed the display function. This was often done by having the display software depend heavily on the data structure used to process or store the results of the analysis.

For example, timing analysis often reveals the portions of the circuit that are too slow. Reviewing this analysis his-

torically has involved examining the schematic and tracing the critical path. However, as described previously, the schematic may have little to do with the designer's specification of the circuit.

This intimate linking of display to analysis causes several problems. First, as described, in conjunction with logic synthesis, displaying the data structure linked with the analysis may not be particularly insightful to the designer. Second, such an intimate linking requires more software development and designer training. If there were N kinds of displays, and M kinds of analysis, and each kind of display could be combined with each kind of analysis, then conventional CAD systems tended to have N*M individual display/analysis programs. This requires the designer to become proficient with a multiplicity of slightly different interfaces as well as requiring the tool supplier to construct all of these tools.

However, modern CAD tools must support the development of chips containing millions of transistors. Designing chips with this many components requires that the CAD tools display and analyze large data structures. Processing large data structures tends to reduce the response time of CAD tools. The conventional technique of tightly coupling the display software to the analysis software helps the CAD system maintain a reasonable response time with the large data structures.

Intimately linking the display tools to the analysis tool data structures poses some problems. First, it tends to require the maker of the CAD tool to produce a large number of products that require support. Second, the variety of such tools tends to introduce variations in the command interface that the designer must use to identify and initiate circuit analysis. Both of these problems lead to frustrated designers and tool builders.

5. Background Conclusion

Using HDL synthesis can simplify the task of circuit design by allowing the designer to specify the required function in an HDL textual description without specifying the details of the circuit implementation. After creating a circuit using synthesis, the designer can use conventional circuit analysis tools to determine characteristics of the final circuit. Conventional analysis will describe such things as the area consumed by different parts of the circuit, or what the longest delay path is through the circuit. Using these analysis results, the designer can then identify which portions of the circuit are problematic. However, because the optimization portion of synthesis often transforms the design substantially, it is difficult, if not impossible, except in certain special cases, to relate specific portions of the final circuit to the HDL source that generated those portions. This inability to trace the circuit analysis results easily to the HDL source represents a substantial barrier for debugging circuits efficiently.

SUMMARY OF THE INVENTION

The present invention provides a method for displaying the results of synthesized circuit analysis visually near the HDL source specification that generated the circuit. Circuit analysis provides information about the characteristics of each portion of the synthesized circuit. The present invention relates the analysis results of each portion of the synthesized circuit to the particular part of the HDL specification that generated that circuit portion. This permits the designer to modify the part of the HDL specification that is responsible for problems identified by circuit analysis. The synthesis process works by translating HDL source code into an initial circuit. Each point in the initial circuit corresponds directly with a particular construct in the HDL source. A final, more efficient circuit is constructed from the initial circuit by logic optimization. Connecting the results of the analysis to the source requires identifying points in the final circuit that be traced directly to the initial circuit. Circuit analysis results corresponding to these invariant points in the circuit can therefore be directly related to the appropriate part of the HDL source, and thus can be displayed near that part.

The present invention provides a method for introducing additional points in the design that remain traceable through the optimization process without requiring reorganization or modification of the HDL source. The present invention provides these additional points, for example, by artificially injecting primary inputs or outputs into the initial circuit, and noting where in the HDL source these points came from.

The present invention provides a method for linking information gleaned from evaluating and analyzing a synthesized circuit to the source code that produced the circuit. The present invention establishes the link by providing a designer with the ability to mark the synthesis source code in the places that the designer wants to be able to debug. The designer marks the source code with a particular text phrase, such as "probe", along with some additional optional information. During translation, the translator generates a circuit the provides the same function as it did without the "probe" statement, but adds additional information or components to the initial circuit that indicate that certain components should not be replaced during optimization. Because those components will not be replaced during optimization, the circuit analysis results corresponding to any unreplaced components that are in the final circuit will be directly and traceably related to those components in the initial circuit. Because those components are traceably related to the source HDL, the results are traceably related to the source HDL, and therefore be displayed near the appropriate portion of the HDL. Allowing for the interjection of unreplaced components by the designer facilitates debugging without rewriting the designer's original hierarchical design or manually backtracking through the optimization process.

In another aspect of the invention, the designer can assign a priority level to each probe to help manage the debugging process. These priority levels could then be used to activate or deactivate selected probes as a group. An activated probe would establish a link through the synthesis process to facilitate debugging. An inactive probe would have no effect on the synthesis process, and would not establish a debugging link. Establishing many links would provide the designer with a large degree of debugging information, but could limit the ability of the synthesis process to provide a good circuit. Establishing too few links may not provide enough guidance to the designer to resolve circuit problems. By selectively activating groups of probes at different times during the debugging process, the designer can analyze different portions of the design without the probes themselves unduly interfering in the process.

By providing a facility for displaying the results of circuit analysis near the HDL that created the circuit, the present invention allows a designer to make more effective use of logic synthesis and reduce the complexity of the circuit debugging process.

An aspect of the present invention provides a method and system for processing requests from designers about the characteristics associated with the logic synthesis source specifying a circuit, and displaying the results of circuit

analysis with a consistent set of display tools that are not intimately tied to the data structure used for the circuit analysis. The present invention achieves this by first observing that the representation of the circuit can be divided into domains characterized by the structure of the information used to represent the design. Then the tool builder can develop domain dependent display tools for examining the state of the design in that domain. In addition, the tool builder can also develop tools that evaluate or analyze the state of the circuit in a particular domain. Display tools showing the circuit structure in one domain can obtain information related to analysis obtained in another domain by the forward and backward linkages provided by this invention.

The designer can inquire about the characteristics related to a specific part of the design by first examining part of the design in one domain with a display tool. This domain is the inquiry domain. After identifying a relevant portion of the design in the inquiry domain, the designer selects a constituent piece of the design to evaluate, and makes an inquiry about that piece. This information constitutes a query. The display tool forwards the identification of the object in the inquiry domain and an identifier indicating the requested analysis or evaluation to the database. The database then determines the domain that would contain the relevant analysis results. If those results do not yet exist, the database invokes the appropriate analysis tool to put those results into the database. Using the linkage established with the HDL-debugging method, the database locates the related object in the analysis domain. From the related object, the appropriate information is passed back to the display tool where the designer can see it displayed appropriately.

One aspect of the present invention provides display tools that are not dependent on the structure of the domain in which the analysis is actually performed. Another aspect of the invention provides analysis tools that are not dependent on the structure of the display domain. Another aspect of the invention is to allow the different display and analysis tools to remain independent from one another.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1**. A flow diagram showing an earlier design process.

FIG. **2**. A flow diagram showing a new design process in accordance with the present invention.

FIG. **3**. A flow diagram showing an earlier synthesis-analysis process.

FIG. **4**. An example of VHDL source with no probes.

FIG. **5**. A parse tree corresponding to the source fragment in FIG. **4**.

FIG. **6**. A circuit arising from the conventional translation of the source fragment in FIG. **4**.

FIG. **7**. An optimized version of the circuit of FIG. **6**.

FIG. **8**. The VHDL source in FIG. **4** with statement probes inserted.

FIG. **9**. Translation of the source in FIG. **8**.

FIG. **10**. An optimized circuit derived from the circuit of FIG. **9**.

FIG. **11**. An example of a display relating information found from analysis of the optimized circuit of FIG. **10** to the source HDL.

FIG. **12**. VHDL source without probes using two process blocks.

FIG. **13**. Conventional translation of the source in FIG. **12** into a circuit.

FIG. **14**. An optimized circuit derived from the circuit of FIG. **13**.

FIG. **15**. An example of a display relating data found from analysis of the optimized circuit of FIG. **10** to the source VHDL showing that information can only be related to the highest level in the description.

FIG. **16**. VHDL source with two block probes installed.

FIG. **17**. A circuit generated by translating the VHDL source of FIG. **16**.

FIG. **18**. The circuit obtained by optimizing the circuit of FIG. **17**.

FIG. **19**. An example of a display relating data found from analysis of the optimized circuit of FIG. **18** to the source VHDL showing information related to the block probes.

FIG. **20**. An alternate method of implementing probes in accordance with the present invention.

FIG. **21**. An second alternate method of implementing probes in accordance with the present invention.

FIG. **22**. Display of the transitive fan-in trace of a particular signal in the source HDL in accordance with the present invention.

FIG. **23**. Display of the primary inputs reached from transitive fan-in trace of a particular signal in the source HDL in accordance with the present invention.

FIG. **24** shows the relationship between the display tools, analysis tools, and the database that maintains the design.

FIG. **25** shows the architecture of FIG. **24** with one display tool and one analysis tool.

FIG. **26** shows a stacked bar graph display of circuit information.

FIG. **27** shows a stacked bar graph display of circuit information showing the relative contribution of one of the sub-blocks in FIG. **26**.

FIG. **28** shows a stacked bar graph display of circuit information showing the relative contribution of one of the sub-blocks in FIG. **27**.

FIG. **29** shows a histogram display of circuit timing information.

FIG. **30** shows a text display of HDL source code and circuit information related to that source code.

FIG. **31** shows a virtual schematic display showing the inputs and outputs associated with a particular part of VHDL source code.

FIG. **32** shows another virtual schematic display tracing the output of the display in FIG. **31**.

FIG. **33** shows another virtual schematic display tracing the output of the display in FIG. **32**.

DETAILED DESCRIPTION OF THE INVENTION

The present invention comprises a novel method for the using the links and establishing new links between an HDL source description of a circuit and the analysis results of the translated and optimized actual circuit that arises from the source description. The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. Various modifications to the preferred embodiment will be readily apparent to those skilled in the art, and the generic principles defined herein

may be applied to other embodiments and applications without departing from the spirit and scope of the invention. Thus, the present invention is not intended to be limited to the embodiment shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

Improved Design and Debugging Process

FIG. **3** shows the general design and debugging process in accordance with the present invention. The designer writes HDL with probe statements **150**. The probe statements identify the places in the resulting circuit that the designer will wish to examine. The designer may not initially know where probes will be required until later in the design process. The probes have no impact on functionality so functional simulation **101** and functional repair **102** proceed as before. The designer also constrains synthesis **103** as before.

Synthesizing with probes **154** differs from conventional synthesis **104**. When the translator encounters a probe statement, the translator interjects information into the netlist to indicate to the optimizer that optimization should not proceed "past" or "through" that particular point. The optimizer then produces a new circuit subject to these constraints. In one embodiment, the probed portions of the HDL source are treated as both primary inputs and primary outputs during translation and optimization.

The circuit analysis step **105** proceeds as before. After analysis, the tool then uses information developed during translation to relate the results of the analysis to the HDL source as indicated by step **120**.

With the information gleaned from the probes, the designer can now identify problems and evaluate solutions that directly change the HDL, as shown in step **121**.

After completely analyzing and debugging the design, the actual circuit can be fabricated as it was before as shown in step **106**.

As described above, optimizers divide an initial circuit's components into two groups: components that can be replaced and those that can't. One method of implementing probes would be to mark the components of the initial circuit attributable to the probed part of the HDL as components that can not be replaced during optimization. For example, the translator could create a primary input and a primary output at the places in the initial circuit corresponding to the probed part of the HDL source.

After optimization, the circuit can then be analyzed with conventional circuit analysis tools to provide an analysis report **904**. For example, the analysis report can indicate the time it takes for a signal to reach various points in the optimized circuit. Analysis could also reveal what portions of the design are not testable or occupy a large amount of area. Because each report provides information linked to the nodes of the optimized circuit, each report therefore provides information about each node that did not change during the optimizer's iterations. Each node that did not change during the optimization process can be traced to a particular place in the HDL source using the information developed in constructing the parse tree.

Examples of Using and Processing Probes

FIG. **4** through FIG. **18** illustrate by example how probes work and how the debugging information could be displayed to the user. The examples use VHDL as the source language. The principles illustrated do not depend on the particular language.

FIG. **4** shows a text editor window **300** containing a VHDL code fragment **400** that does not contain any probe statements. The code fragment shown in FIG. **4** is repeated below:

```
if (C and B) then
    Z <= not(A or B);
else
    Z <= not(B);
end if;
```

FIG. **5** shows a graphical representation of the parse tree constructed while translating the source code in FIG. **4**. Link **1100** is the connection that this fragment has with the rest of the VHDL description. The "if" statement in VHDL has three parts: a condition; a VHDL statement to process when the condition is true; and a VHDL statement to process when the condition is false. The condition is dealt with in the tree descending from node **1001**. The true condition is handled by the tree descending from node **1004**. The false condition is handled by the tree descending from node **1010**. The assignments represented by nodes **1004** and **1010** are used to link the signal values represented by node **1005** and node **1011** to their functions represented in the trees descending from nodes **1006** and **1012** respectively.

Without a probe statement, the VHDL fragment would translate into the circuit of FIG. **6**. Inputs A, B, and C are schematically represented by the connectors **200**, **201**, and **202**. The "if" statement would translate into multiplexor **231**. The condition "(C and B)" would translate into and gate **232**. The "true" condition would translate into nor gate **233** while the "false" condition would translate into invertor **230**.

FIG. **7** shows a circuit optimized from the digital circuit of FIG. **6**. In particular, the logic function that this code fragment really performs is not(B). At this point, without probes in the conventional synthesis design process, the designer can not obtain much information about the internal timing information descending from the fragment. For example, if the designer needed to know when the value of not(A or B) was computed to help analyze some other aspect of the design, then the designer would not be able to deduce that information from the resulting analysis.

FIG. **8** shows how a probe statement **401** could be inserted into the source description. The code fragment is repeated below:

```
if (C and B) then
    Z <= not(A or B); --Synopsys probe_statement
else
    Z <= not(B);
end if;
```

In VHDL, "--" begins a comment. The word "Synopsys" immediately after the "--" indicates that this is not an ordinary comment, but rather a directive to the translator or other part of the computer aided design tool environment. The word "probe_statement" indicates that this particular VHDL statement should be processed so that it will be possible to relate analysis information to this point in the circuit.

FIG. **9** shows a circuit that a translator could produce from the code fragment shown in FIG. **8** with the probe statement. The parse tree produced with the probe statement is the same as before, namely the tree of FIG. **5**. However, the probe statement will cause the signal represented by node **1005** to behave as both a primary output and a primary input.

In creating the circuit of FIG. **9** from the parse tree, the translator could add temporary input **203** and a new temporary output **221**. In addition to synthesizing this circuit, the translator connects the new temporary input to the new output at a higher level in the net list produced from translating the whole specification.

FIG. **10** shows the circuit of FIG. **9** after optimization. The optimizer is not permitted to optimize circuits past the boundaries established by the probe statement. This means that nor gate **233** of FIG. **9** would be transformed into nor gate **253** of the optimized circuit. The nor gates are not necessarily identical because additional details may be specified about the gates during the design compilation process. Those details are not relevant to the implementation of probes.

Because the logic optimization process left temporary input **203** and temporary output **221** alone, and those points correspond to a particular point in the HDL circuit, any analytic result related to temporary input **203** or temporary output **221** can be identified with the probe statement **401** in the HDL. FIG. **11** shows how timing information could be related back to the HDL in a special text window **301**. For example, suppose a critical path analysis tool determined that it took 1.0 nanoseconds to produce temporary output **221** of FIG. **10** after a clock edge arrived at a flip-flop somewhere else in the circuit. By using the fact that temporary output **221** came from this line of the source, the timing result **500** can be displayed next to the appropriate line of the output.

FIG. **12** through FIG. **19** show another way to use probes to evaluate the performance of blocks of HDL code. FIG. **12** shows a text window with an HDL entity described. This text has no probe statements inserted, and the code is repeated below.

```
entity interrupt_controller is
    port(new_request : in bit_vector(3 downto 1);
         current_level: in bit_vector(1 downto 0);
         should_service: out bit);
end;
architecture synthesizable of interrupt_controller is
    signal new_level: bit_vector(1 downto 0);
begin
    decode: process(new_request)
    begin
        if(new_request(3) = '1') then
            new_level <= "11";
        elsif(new_request(2) = '1') then
            new_level <= "10";
        elsif(new_request(1) = '1') then
            new_level <= "01";
        else
            new_level <= "00";
        end if;
    end process;
    compare: process(current_level, new_level)
    begin
        if(new_level(1) > current_level(i)) then
            should_service <= '1';
        elsif(new_level(1) < current_level(1)) then
            should_service <= '0';
        elsif(new_level(0) > current_level(_)) then
            should_service <= '1';
        else
            should_service <= '0';
        end if;
    end process;
end;
```

This VHDL source code computes whether a new interrupt should be serviced. It determines what the new level of the interrupt is by determining what input line is asserted, and comparing the interrupt priority level associated with that with the level of the currently pending interrupt. If the new level is higher, then the output should_service is set high, and otherwise it is set low.

FIG. **13** shows the circuit resulting from translating the VHDL source. FIG. **14** shows the circuit that results from optimizing the circuit in FIG. **13**.

FIG. **15** shows a special text window that summarizes some of the performance information about the circuit. The analysis tool can provide information about the design as viewed from the inputs. An analysis tool could provide an estimate of the area of the design by counting gates or compute the longest delay through the entire design. The designer might conclude that this circuit is too big or too slow. Because all of the inner details of the design have been optimized, it is not realistic for the designer to examine the schematic in FIG. **14** manually and determine whether the decoding function or the comparing function is too big or too slow.

To determine where the problems lie, the designer could insert block probes textually near the definition of the decoding and comparing processes, as shown in the text window of FIG. **16**. This would probe all signals entering or leaving the sequence of HDL code delineated by the begin block and end block statements. When translated, the probed HDL would become the circuit of FIG. **17**. The translator would create temporary inputs **2010** and **2011**, and temporary outputs **2000** and **2001**, much as it did with the statement probe.

The optimizer transforms the circuit of FIG. **17** into the circuit of FIG. **18**. This allows the analysis tools to compute timing and area characteristics of both parts of the circuit. A special purpose display tool can then display, for example, timing and area analysis, as shown in FIG. **19**. In this example, the decoding circuit is approximately the same size and delay as the comparator circuit.

The block probes used in FIGS. **16** through **20** illustrate a mechanism to select many parts of the HDL source for probing without typing a text statement probe command for each line of HDL. In particular, the block probes in the previous example prohibited the optimizer from eliminating the inputs and outputs of the block. Another kind of block probe that might be useful is one that places a probe on every signal within the block. This will significantly limit the optimizer's ability to improve the circuit within the block, but it will give the designer the maximum amount of information about how the HDL source was translated. A third kind of block probe could place a probe on every statement in the block. A fourth kind of block probe could place a probe on every signal driving multiplexor control inputs within the block. Multiplexors are generated in HDL translation to implement "if" and "case" constructs. Probing the control inputs to these multiplexors would provide a designer with much insight into the circuit behavior.

As described previously, the circuit produced when probes are used can be different from the circuit that occurs when probes are not used. Because probes interfere with the ability of the optimizer to produce higher quality circuits, the designer generally should only use them when the designer needs to gather particular information. During the debugging process, a designer may insert many probe statements into the HDL source at various times to discover the characteristics of different parts of the circuit. One method of managing the number and location of active probes would be to let the designer specify a number with every probe statement that would be the statement's priority. When invoking the optimizer, the designer could then specify a

optimization priority. The optimizer would then eliminate every probe with a priority greater than the invoked priority. This would have the effect of permitting the designer to only use the most important probes without having to textually remove the lesser probes.

Alternate Methods for Probing

FIG. **20** shows an alternate method of having the translator implement probes. The circuit in FIG. **20** would come from the VHDL code fragment of FIG. **11**. Instead of creating both a new input and a new output, only a new output **210** is created.

FIG. **21** shows another method of having the translator implement probes. The circuit in FIG. **21** would come from the VHDL code fragment of FIG. **11**. Here, a new part **270** is inserted into the design. This part would need to have an additional attribute associated with it to indicate to the optimizer that it was not to be eliminated.

Another method of implementing probes requires attaching an attribute to the net connecting gates together. For example, in processing a probe statement, the translator could create the circuit in FIG. **6**, but add information to net **280** that net **280** could not be removed during the optimization process.

Tracing Transitive Fan-In and Fan-Out in the Source HDL

During debugging, the designer may need to consider the impact that a change in one part of the design would have other parts of the design. If the designer is considering changing a particular function in HDL, the designer would find it useful to identify all of the inputs to that function or all of the outputs to that function. The collection of all of the points in a circuit leading to a particular point is referred to as the Transitive Fan-In of that point. The collection of all of the points in a circuit that depend on the value of a particular point is the transitive fan-out. While tracing all of the inputs (or outputs) of a particular part of the source HDL is a difficult task using only the HDL source, using the direct correspondence between the HDL and the initial circuit formed during translation makes it possible to highlight the inputs (or outputs) in the source HDL.

For example, a designer might be interested in finding all of the computations that feed into the multi-bit signal new_level in the VHDL source of FIG. **12**. The designer could select the signal new_level with a mouse and enter a trace command with keystrokes or mouse clicks. A tracing program would trace the desired signal through all computations leading to primary inputs. As the tracing program traces signals back through the initial circuit, it identifies those points corresponding directly to the HDL source, and makes it possible for the text editor to highlight the corresponding HDL source. FIG. **22** provides an example of the text that would be highlighted during such a trace.

Alternatively, the designer might be interested only in the primary inputs or registers that lead to the indicated signal. FIG. **23** shows the result of tracing new_level back to the primary inputs.

System Architecture

FIG. **24** illustrates the architectural relationship between the display tools **111**, **112**, **113**, **114**, **115**, **116**, **117**, **118**, and **119** the analysis tools **130**, **131**, **132**, **133**, **134**, **135**, **136**, **137**, and **138** and the database **125**. The designer **520** interacts with the display tools by seeing information on a screen and providing information to the tool by keyboard and mouse. The database **125** contains different representations of the circuit design under development. These representations are grouped into domains **1500**, **1510**, **1520**, **1530**, and **1540**. The analysis tools **130**, **131**, **132**, **133**, **134**, **135**, **136**, and **137** interact with the design information in the various domains to summarize and evaluate the design.

Design Representation: Domains

Providing useful information to the designer about the current state of a circuit design requires an explanation of how different aspects of a designs are stored in a CAD system. Previous sections described the process that a designer goes through to create and debug a design using logic synthesis. In going through the design process, the designer, through the CAD system, manipulated and transformed digital data having some information about the circuit into other digital data that had different information about the circuit. This section explains how the information describing all of the aspects of a design is organized into domains.

A domain is a collection of the design data that contains common structural characteristics. Each domain represents a particular level of abstraction of the design information. In the CAD system architecture of FIG. **24**, the design can be divided into a source domain **1500**, a generic technology domain **1510**, which is also known as a G-Tech domain, a gate domain **1520**, a layout domain **1530**, and possibly other domains **1540**. The design data in one domain can be the result of a transformation of design data from another domain using design tools, such as a logic synthesizer, and libraries of components.

The source domain **1500** contains the HDL source files that the designer creates in step **100** of FIG. **1** or step **150** of FIG. **3**. It also contains the parse trees and symbol tables generated during the translation step of logic synthesis. Here, this portion of the design representation only contains information about the desired function of the circuit without reference to circuit topology or technology.

The generic technology domain **1510** contains the initial circuit that arises from the translation step of the synthesis process, as shown in step **104** of FIG. **1** or step **154** of FIG. **3**. Data stored in the generic technology domain **1510** contains information about the topology of the circuit, but does not have information about the specific technology to be used.

The gate domain **1520** contains the final circuit that arises after the optimization step of the synthesis process, as shown by link **502**. The debugging method previously described establishes the reverse link **505**. Like the generic technology domain **1510**, the data in the gate domain **1520** contains information about how components are connected together. However, in the gate domain, a particular technology is specified, thus providing information about the physical characteristics of the components used to implement the desired function. It is in this domain that preliminary timing, area, power, testability, and other calculations of step **105** of FIG. **1** and FIG. **3** can be made.

The layout domain **1530** contains information about the geometric placement of the components on the chip substrate and the connections between the components. The design information in the layout domain **1530** is obtained from the design information in the gate domain **1520** by using placement and routing tools.

It would also be possible to have additional domains, as shown by other domains **1540**.

Objects within a Domain

Conceptually, the design information within a domain can be thought of as a collection of interconnected objects, with the objects and the connections possessing certain characteristics. For example, in the source domain, the objects could include the text of the HDL source code or the nodes of the parse tree constructed from the source code or the entries in the symbol table. In the gate domain **1520**, the objects could include the individual gates or other library

parts or the connections between them. In any case, database **125** maintains the relationships between the objects within the domain. The debugging method presented discussed earlier shows how to establish a link between a group of objects in one domain and a group of related objects in another domain.

Display Tools

The information specifying the design is actually represented in binary form within a computer system. For the human designer **520** to develop and debug the design, information about that design must be presented in a form that can be understood by the human, such as by a visual display on a monitor. In addition, the designer **520** must also have a way to manipulate the design information, such as by keyboard and mouse. Display tools **111**, **112**, **113**, **114**, **115**, **116**, **117**, **118**, and **119** provide a mechanism for that interaction.

However, circuit design information is particularly complicated, and efficiently allowing the designer **520** to control and analyze the design requires that the display tools take advantage of the structure of the design data. One way to take advantage of the structure is to design particular display tools to communicate with a particular domain in the database. For example, as shown in FIG. **24**, display tools **111**, **112**, and **113** communicate with the source domain **1500**. Display tools **114** and **115** communicate with the G-Tech domain **1510**. Display tools **116** and **117** communicate with the gate domain **1520** while display tools **118** and **119** communicate with the layout domain.

A display tool interacts with its related domain by communicating messages back and forth. These messages involve specifying an object in the domain and some related action to perform on that object. The display tool is customized to interact with a particular domain in the sense that it is constructed to process objects in that domain. In particular, a display tool needs to be able to display an object and have a list available of operations that can be performed on that object, although the display tool does not perform those operations. The database, in contrast, needs to be able to provide the display tool with objects and process requests associated with objects.

Analysis Tools

Analysis tools are used to process information contained within a domain. In general terms, there are two kinds of analysis that one might want to perform in a given domain. The more conventional kind of analysis tool takes one or more objects in a domain, and computes characteristics associated with those objects. For example, in the gate domain, a timing analyzer is used to compute the delay times and slacks to various nodes in the circuit. In the analysis sense, the timing analyzer is determining a certain characteristic, for example, the time that the data signal will become valid after a clock edge, of certain objects in the domain, namely a connection between gates.

Another kind of analysis involves summarizing the characteristics associated with a particular group of objects. For example, estimating the total area in a circuit at the gate level involves summing the area associated with each component and connection.

For an analysis tools to summarize characteristics of objects, the analysis tools requires information about the structure of the objects and the connections between them. Therefore, analysis tools are associated with a particular domain. For example, in FIG. **24**, analysis tools **130** and **131** deal with the source domain **1500**, while analysis tools **132** and **133** deal with the G-Tech domain **1510**. Analysis tools **134**, **135**, **136**, and **137** deal with the gate domain **1520**, while analysis tool **138** interacts with the layout domain.

In general terms, analysis tools communicate with the database **125** by messages. The database provides the analysis tool with one or more objects, and the analysis tool responds by setting one or more characteristics of those objects or returning a summary of the characteristics of those objects, or both.

Using the Architecture for Debugging with Reference to the Source Domain

This section explains how a designer uses the CAD system architecture to relate characteristics of the design found in one domain to aspects of the design found in another domain. FIG. **25** shows a simplified view of the architecture of FIG. **24** involving only one display tool and one analysis tool. For this example, the design in question is assumed to be complete to the gate domain **1520**.

The designer **520** of FIG. **25** begins the pursuit of design insight by first obtaining a display that engages in the debugging process. Next, he obtains a visual representation of an aspect of the design stored, for example, in the source domain **1500** using display tool **109**. At the time the display tool **109** begins executing, the database **125** provides the display tool **109** with a list of analysis requests that the designer **520** can request. The display tool **109** could display the text of an HDL source file. The HDL text is an object in the source domain **1500**. The processing example described in this section corresponds to the example used in FIG. **12** through FIG. **19**.

The designer **520** then selects an object to evaluate by selecting some text, such as the decode: process statement in the text of FIG. **16**, and allowing the database **125** to identify the parse tree nodes corresponding to the selected text. Alternatively, to improve performance, the database **125** could transfer data to the display tool to allow the display tool to identify the parse tree node. A good technique for relating particular pieces of text with corresponding parts of a parse tree is described in a co-pending application by Gregory entitled "Method and Apparatus for Context Sensitive Displays", filed on Jun. 3, 1994 with express mail certificate TB596163183US, which is hereby incorporated by reference. The parse nodes are also objects in the source domain.

The designer **520** also selects a type of analysis to be performed from the available choices. One approach is that the designer **520** selects it from a menu or a push button. Another approach would be to have the designer **520** select it before selecting the text. In this example, assume that the designer **520** asks for an estimate of the area of the decode process.

The parse node and the desired analysis form a query that is sent to the database **125**. Because area requires technology specific information, the database **125** can identify that the required information is in the gate domain **1520**. In essence, the database **125** then needs to compute the area corresponding to the identified parse node. The database does this by identifying the gates in the gate domain **1520** that correspond to that parse node. One way that this can be done is to use the techniques described earlier using hierarchy and probes. In this example, the designer **520** wanted to know what the area of one of the processes is, and the designer **520** used probe points to segregate the design to permit this.

Given that the database **125** has identified the relevant gates, it then needs to compute the total area. In the gate domain **1520**, each gate is an object with an associated area characteristic. In this situation, the database could calculate the total area with a summarizing type of analysis tool **139** that sums up the area of each gate in a list. In other situations, it may be necessary to perform a sequence of

simpler analysis operations on parts of the data structure to deduce the correct result. The analysis request from the display tool could also include a list of such instructions. Analysis tool **139** then produces a number that is handed back to database **125** which then sends to display tool **109** which can display the result directly or use it to modify the display characteristics.

Display Tools

The flexibility of the architecture shown in FIG. **24** permits additional display tools that relate circuit analysis information about parts of the circuit to the source text that generates those parts. FIG. **26** through FIG. **33** show novel display techniques for displaying circuit analysis data. FIGS. **26** through **33** show the display tools displaying an AMD2910A design. The source and circuit is described in Introduction to HDL-Based Design Using VHDL, by Steve Carlson, published in 1991, which is hereby incorporated by reference. This book is available from Synopsys, Inc., 700 East Middlefield Road, Mountain View, Calif. 94043-4033.

Stacked Bar Graph Display

FIG. **26** shows a stacked bar graph displaying information about the relative contributions of parts of the synthesis source. Often a design written in an HDL is described hierarchically, with higher level modules containing lower level modules. At a particular level in the hierarchy, the designer might want to know the characteristics of the modules visible at that level. The stacked bar graph of FIG. **26** shows relative areas associated with different parts of the design. At the highest level of the AMD 2910A, there are five functional sub-blocks: CNTL_BLK, MUX_OUT_BLK, REG_BLK, UPC_BLK, and STACK_BLK. The names of these blocks are shown in the object list area **2610** of window **2600**. The total measured area is displayed at the bottom of the window. The total area of the circuit is shown as 1964.0 gates. In this example, the characteristic is area. However, other characteristics such as power and time can be similarly displayed.

Each of the sub-blocks has a measured characteristic which is shown by text statements **2680** through **2684**. For example, CNTL_BLK uses 74.00 gates, which is 3.8% of the total as shown by statement **2680**. MUX_OUT_BLK occupies 148.00 gates, which is 7.5% of the total as shown by statement **2681**. REG_BLK occupies 225.00 gates, which is 11.5% of the total as shown by statement **2682**. UPC_BLK occupies **237.00** gates, which is 12.1% of the total as shown by statement **2683**. STACK_BLK occupies 1280.00 gates, which is 65.2% of the total as shown by statement **2684**. A stacked bar graph is constructed by drawing a graphical box corresponding to each functional sub-block with the size of the box proportional to the percentage of the sub-block's characteristic to the total characteristic. This is shown with boxes **2630** through **2634**.

Importantly, the stacked bar graph display of FIG. **26** can be constructed without reference to the physical nature of the particular characteristic. Therefore, power or timing can be displayed as easily as area. The database need only transfer the names of objects and the numerical value associated with those objects to the display tool.

Furthermore, each box, such as box **2630** through box **2634**, forming part of the stacked bar graph can also be a selectable button. The user can "push" the button and gain information about the sub-block associated with the box. FIG. **27** shows the result of the user selecting the sub-block MUX_OUT_BLK box by selecting box **2631**. Here, the sub-block MUX_OUT_BLK itself contains two sub-blocks, OUT_BLK and MUX_BLK. The total measured characteristic **2620** changes to 148.00 to reflect the size of MUX_OUT_BLK. The sub-block OUT_BLK has 49.00 gates representing 33.1% of the area of MUX_OUT_BLK, as indicated by statement **2780**. This information is also shown graphically by box **2730**. The sub-block MUX_BLK has 99.00 gates representing 66.9% of the area of MUX_OUT_BLK, as indicated by statement **2781**. This information is also shown graphically by box **2731**. In addition, the window also shows the current location in the hierarchy with a path statement **2705**. In addition, statements such as statement **2780** could also act as buttons to change levels.

FIG. **28** shows the information displayed if the designer selects MUX_BLK to see how the 99.00 gates are allocated.

Histogram Display

The histogram display of FIG. **29** can provide a designer with information about the extent of problems currently in the design. For example, one aspect of the performance of a digital circuit is the delay along any path from the output of one flip-flop to the input of another. Once a designer specifies the clock waveforms, a timing verifier can determine arrival and required times throughout the clocked circuit. At any point in the circuit, the arrival time can be determined relative to a clock edge by measuring the longest path from a register affected by the clock to the point within the circuit. Similarly, required times may be computed relative to a clock edge by measuring the longest path from the point in the circuit to a register affected by the clock. Once the relationships between all clocks and all clock waveforms are specified, the timing verifier can determine the worst slack for each point within the circuit by subtracting arrival from required times for each possible combination of relevant clock edges. The smallest, or most negative result can be considered the worst slack for that point in the circuit. The Design Compiler Reference Manual V3.1a from Synopsys contains more information about timing analysis, and is hereby incorporated by reference. If any node has a negative slack time, then the timing goal has not been met. If only a few nodes are have negative slacks, or the negative slacks are close to zero, then the designer may merely have a relatively small problem that can be fixed by tuning a small portion of the design. However, if many nodes have negative slacks or the slack times are large in magnitude, then the designer may be facing a substantial design problem. The histogram tool of FIG. **29** can provide guidance on the extent of a problem facing a designer.

A histogram tool provides a mechanism for displaying the distribution of a particular numerical characteristic of the circuit and allowing a user to see a list of objects having that characteristic. The example in FIG. **29** displays timing analysis for the AMD **2910**. The database uses a timing analyzer and a targeted clock period to compute the slack times on every net. The circuit nodes with similar slack times are grouped into bins, and counted. Histogram-list window **2900** contains two sub-windows: the histogram window **2920** and the list window **2910**. The histogram window **2920** contains bars **2930** through **2937** with one bar per bin. The height of the bars indicates the number of nets that fall into that bin. If the user selects one of the bars, the list window **2910** shows the list of names that identify the nets that are in that bin. Individual items in the list display can be selected, as indicated by selected item **2915**. This allows the designer to use other display tools to gain more information about the selected item.

Text Display

FIG. **30** shows the text display of HDL source code that annotates that source code with additional information. Text display window **3000** contains three smaller windows. Text window **3010** contains the source text. Select window **3040**

shows circuit information related to text that has been selected. Cursor window **3030** shows circuit information related to text that is under the cursor. Column report **3060** shows circuit information associated with each line of the text. Selecting text can be done with the usual window based text selection mechanisms. For example, the designer could move a cursor to the relevant portion of the screen and push a button. The text window **3010** constructs a text box **3020** around the object that the cursor is pointing at. One fast method of determining the limits of cursor text box **3020** would be to use the parse tree representation described in co-pending application by Gregory entitled "Method and Apparatus for Context Sensitive Displays", filed on Jun. 3, 1994 with express mail certificate TB596163183US.

Here, cursor window **3030** is showing an estimate of the circuit area associated with the object under the cursor. The phrase "gtech Area=6" indicates that the implementation of the comparison function performed by condition "PSH_PTR>STK_LOW" indicated by cursor text box **3020** requires 6 area units in the generic technology domain in the database. Cursor window **3030** could display other characteristics associated with the text pointed to by the cursor.

Select window **3040** shows information associated with selected text. Here, the size and font of the selected text **1050** is changed. One fast method of determining the limits of selected text **1050** would be to use the parse tree representation described in co-pending application by Gregory entitled "Method and Apparatus for Context Sensitive Displays", filed on Jun. 3, 1994 with express mail certificate TB596163183US. In this example, the select window **3040** shows detailed information about the HDL construct MEM [PSH_PTR] **3050**. Statement **3080** shows the type of HDL construct that the selected object is—in this case, the construct is an array index. Statement **3083** shows the estimated area of the construct in the G-Tech domain **1510**, here 530 area units. Statement **3084** shows the length of the longest path in levels of logic from a register to the gates that implement the construct, here 18 gates. Statement **3081** shows the parse tree node number. A detailed list **3082** shows the netlist components in the G-Tech domain implementing the construct **3050**. For each component in this list, the following information is displayed: the component's netlist instance name **3087**, the type of netlist component **3088** (reference name), its contribution to the total area estimate **3089**, and the class of netlist component **3090** e.g. cell, pin, net, or port. Other information could be displayed at the designer's option.

Column report **3060** shows information associated with each line. Here, the column report is showing the area associated with the HDL constructs on each line.

Virtual Schematic Display

Part of the circuit debugging process involves tracing the drivers and driven or, inputs and outputs of specific circuitry. The virtual schematic display shown in FIG. **31** provides the designer with the ability to find the HDL source that provides inputs to and takes outputs from a particular point in the HDL source. The virtual schematic display has a virtual schematic window **3100** which has three window regions: an input region **3110**, a current region **3120**, and an output region **3130**. The designer uses the cursor to indicate selected text **3150** in the current region. This selected text **3150** corresponds to a circuit object **3151** (not shown) in the database. Circuit object **3151** has inputs and outputs. The database then links the input region **3110** to those portions of the text of the synthesis source that show where the inputs of circuit object **3151** originated. Here, the input "DATA" comes from an input to the module MULTIPLEXOR as indicated by input argument **3145**. The database also links the output region **3130** to those portions of the text of the synthesis source that show where the outputs of circuit object **3151** go to. Here, as indicated by output argument **3155**, output "Z" goes to the output of module MULTIPLEXOR.

By clicking in the output region, the designer can trace the output further. FIG. **32** shows the changes that occur in the regions. The text of the output region now moves to the current region **3120**. The text of the output region changes to show synthesis source text corresponding to objects driven by output argument **3155**. Here, output argument **3155** drives another output at the next level module boundary, as shown by output argument **3156**. Input region **3110** changes to show all of the places in the source text that set or define the selected output argument **3155**. Here, there are five text sources for output argument **3155** as shown in windows **3271**, **3272**, **3273**, **3274**, and **3275**. The originally selected statement **3150** is shown again in window **3272**.

An additional input comes from the MULTIPLEXOR input argument SEL as shown in window **3271**. Z is also takes on values at different points in a case statement as shown in windows **3272**, **3273**, **3274**, and **3275**.

FIG. **33** shows the results of pursuing the output argument **3156**. Here, the high level module definition appears in the current region **3120** while the input region **3110** displays the module interface shown in the current region of FIG. **32**.

What is claimed is:

1. A method, comprising:

translating a synthesis source text file to an initial circuit including one or more parts connected together with nets;

identifying each part of said initial circuit with the portion of said synthesis source text file that created said part of said initial circuit;

optimize said initial circuit to produce a final circuit containing one or more parts connected together with nets;

analyzing said final circuit to determine characteristics associated with said final circuit's parts and with said final circuit's nets;

identifying said final circuit's part's that correspond directly with said initial circuit's initial parts;

identifying said final circuit's nets that correspond directly with said initial circuit's nets; and

displaying said characteristics associated with those said final circuit's nets and parts that correspond directly with said initial circuit's nets and parts near said portions of said synthesis source text file that created said corresponding initial circuit parts and nets.

2. The method of claim **1** wherein said source text file includes directives indicating the location of probes and wherein said translating step include generating initial circuit parts or initial circuit nets that will correspond directly with said final circuit parts or final circuit nets.

* * * * *

CERTIFICATE OF SERVICE

I hereby certify that on December 22, 2014, the foregoing Corrected Principal and Response Brief with Addendum of Cross-Appellant Mentor Graphics Corporation was electronically filed with the Clerk of the Court for the United States Court of Appeals for the Federal Circuit using the appellate CM/ECF system. All participants in the case are registered CM/ECF users and service will be accomplished by the appellate CM/ECF system.

Dated: December 22, 2014

Respectfully submitted,

/s/ Christopher L. McKee
Christopher L. McKee
BANNER & WITCOFF, LTD.
Suite 1200
1100 13th Street, NW 20005
Telephone: (202) 824-3000

Attorney for Cross-Appellant Mentor Graphics Corporation

CERTIFICATE OF COMPLIANCE UNDER FEDERAL RULES OF APPELLATE PROCEDURE 28.1(e)(2)(B)(i)

Counsel for Mentor Graphics Corporation certifies that the brief contained herein has a proportionally spaced 14-point typeface, and contains 15,044 words, based on the "Word Count" feature of Word 2010, including footnotes and endnotes. Pursuant to Federal Rules of Appellate Procedure 28.1(e)(3) and 32(a)(7)(C) and Federal Circuit Rule 32(b), this word count does not include the words contained in the Certificate of Interest, Table of Contents, Table of Authorities, Abbreviations, and Statement of Related Cases.

Dated: December 17, 2014

Respectfully submitted,

/s/ Christopher L. McKee
Christopher L. McKee
Bradley C. Wright
Michael S. Cuviello
BANNER & WITCOFF, LTD.
Suite 1200
1100 13th Street, NW 20005
Telephone: (202) 824-3000

Attorney for Cross-Appellant Mentor Graphics Corporation